Haskell Communities and Activities Report

http://tinyurl.com/haskcar

Twenty-Second Edition — May 2012

Janis Voigtländer (ed.)

Andreas Abel, Iain Alexander, Heinrich Apfelmus
Emil Axelsson, Christiaan Baaij, Doug Beardsley
Jean-Philippe Bernardy, Mario Blažević, Gwern Branwen
Joachim Breitner, Björn Buckwalter, Douglas Burke
Carlos Camarão, Erik de Castro Lopo, Roman Cheplyaka
Olaf Chitil, Duncan Coutts, Jason Dagit
Nils Anders Danielsson, Romain Demeyer, James Deng
Dominique Devriese, Daniel Díaz, Atze Dijkstra
Facundo Dominguez, Adam Drake, Andy Georges
Patai Gergely, Jürgen Giesl, Brett G. Giles
Andy Gill, George Giorgidze, Torsten Grust
Jurriaan Hage, Bastiaan Heeren, PÁLI Gábor János
Guillaume Hoffmann, Csaba Hruska, Oleg Kiselyov
Michal Konečný, Eric Kow, Ben Lippmeier
Andres Löh, Hans-Wolfgang Loidl, Rita Loogen
Ian Lynagh, Christian Maeder, José Pedro Magalhães
Ketil Malde, Antonio Mamani, Alp Mestanogullari
Simon Michael, Arie Middelkoop, Dino Morelli
JP Moresmau, Ben Moseley, Takayuki Muranushi
Jürgen Nicklisch-Franken, Rishiyur Nikhil, David M. Peixotto
Jens Petersen, Simon Peyton Jones, Dan Popa
David Sabel, Uwe Schmidt, Martijn Schrage
Tom Schrijvers, Andrew G. Seniuk, Jeremy Shaw
Christian Höner zu Siederdissen, Michael Snoyman, Jan Stolarek
Martin Sulzmann, Doaitse Swierstra, Henning Thielemann
Simon Thompson, Sergei Trofimovich, Marcos Viera
Janis Voigtländer, David Waern, Daniel Wagner
Greg Weber, Kazu Yamamoto, Edward Z. Yang
Brent Yorgey

Preface

This is the 22nd 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 usual mailing lists 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.org

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

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

Since the last community report, the following has happened:

haskell.org incorporation

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

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

Assets

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

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

The haskell.org income for 2011 includes:

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

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

Haskell.org has the following server assets:

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

General

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

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

1.2 Haskellers

Report by:	Michael Snoyman
Status:	experimental

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

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

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

Further reading

http://www.haskellers.com/

2 Books, Articles, Tutorials

2.1 Haskell: the craft of functional programming, 3rd edition

Report by: Simon Thompson

The third edition of one of the leading textbooks for beginning Haskell programmers is thoroughly revised throughout. New material includes thorough coverage of property-based testing using QuickCheck and an additional chapter on domain-specific languages as well as a variety of new examples and case studies, including simple games.

Existing material has been expanded and re-ordered, so that some concepts — such as simple data types and input/output — are presented at an earlier stage. The running example of Pictures is now implemented using web browser graphics as well as lists of strings.

The book uses GHCi, the interactive version of the Glasgow Haskell Compiler, as its implementation of choice. It has also been revised to include material about the Haskell Platform, and the Hackage online database of Haskell libraries. In particular, readers are given detailed guidance about how to find their way around what is available in these systems.

Publication details:

- Published by Addison Wesley, 2011. ISBN 0201882957.

Book website:

- http://www.haskellcraft.com

Solutions for *bona fide* instructors are available from the Pearson website http://www.pearsoned.co.uk/HigherEducation/Booksby/Thompson/

2.2 In Japanese: Learn You a Haskell for Great Good!

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

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

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

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

Publication details:

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

Book website:

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

Further reading

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

2.3 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 editorship of The Monad Reader has passed over from Brent Yorgey to Edward Z. Yang. A mini-issue is currently in the works.

Further reading

http://themonadreader.wordpress.com/

2.4 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 two additions:

Type-level call-by-value lambda-calculator in three lines

The article describes a *type-level* interpreter for the call-by-value lambda-calculus with booleans, natural numbers, and case discrimination. Its terms are Haskell types. Using functional dependencies for type-level reductions is well-known. Missing before was the encoding abstractions with named arguments and closures.

The core of the interpreter indeed takes only three lines

$$\begin{aligned}&\textbf{instance } E\ (F\ x)\ (F\ x)\\&\textbf{instance } (E\ y\ y', A\ (F\ x)\ y'\ r)\\&\qquad\Rightarrow E\ ((F\ x) :< y)\ r\\&\textbf{instance } (E\ (x :< y)\ r', E\ (r' :< z)\ r)\\&\qquad\Rightarrow E\ ((x :< y) :< z)\ r\end{aligned}$$

The first line says that abstractions evaluate to themselves, the second line executes the redex, and the third recurs to find it. The representation of abstractions is apparent from the expression for the `S` combinator, which again takes three lines

$$\begin{aligned}&\textbf{instance } A\ (F\ CombS)\ f\ (F\ (CombS, f))\\&\textbf{instance } A\ (F\ (CombS, f))\ g\ (F\ (CombS, f, g))\\&\textbf{instance } E\ (f :< x :< (g :< x))\ r\\&\qquad\Rightarrow A\ (F\ (CombS, f, g))\ x\ r\end{aligned}$$

The instances of the type class `A f x r` define the result `r` of applying `f` to `x`. The last line shows the familiar lambda-expression for `S`, written with the familiar variable names `f`, `g`, and `x`. The other two lines build the closure 'record'. The closure-conversion is indeed the trick. The second trick is taking advantage of the instance selection mechanism. When the type checker selects a type-class instance, the type checker instantiates it, substituting for the type variables in the instance head. The type checker thus does the fundamental operation of substitution, which underlies beta-reduction.

The article shows many examples, of the fixpoint combinator, Fibonacci function, and S and K combinators.

http://okmij.org/ftp/Computation/lambda-calc.html#haskell-type-level

Applications of computable types

The follow-up article describes several applications of computable types, to ascribe signatures to terms and to drive the selection of overloaded functions. One example computes a complex XML type and instantiates the `read` function to read the trees of only that shape.

A telling example of the power of the approach is the ability to use not only `(a->)` but also `(->a)` as an unary type function. The former is just `(->) a`. The latter was considered impossible. The type-level lambda-calculus interpreter helps, letting us write `(->a)` almost literally as `(flip (->) a)`. For example, we can express the type `(((Int -> Bool) -> Bool) ... -> Bool) -> Bool`, with `n` nested arrows as `E (F Ntimes :< (F Flip :< (F (ATC2 (->))) :< Bool) :< Int) n` where the higher-order type function `NTimes` is the right fold on type-level numerals.

http://okmij.org/ftp/Haskell/types.html#computable-types

2.5 Yet Another Lambda Blog

Report by: Jan Stolarek
Status: ongoing

Yet Another Lambda Blog is a new blog about functional programming aimed at beginners. It focuses on practical aspects of programming in Haskell, but there are other topics as well: book reviews, links to interesting internet resources and Scheme programming. New posts appear once or twice a week.

Further reading

http://ics.p.lodz.pl/~stolarek/blog/

3 Implementations

3.1 Haskell Platform

Report by: Duncan Coutts

Background

The Haskell Platform (HP) is the name of the "blessed" set of libraries and tools on which to build further Haskell libraries and applications. It takes a core selection of packages from the more than 3500 on Hackage (→ 6.6.1). It is intended to provide a comprehensive, stable, and quality tested base for Haskell projects to work from.

Historically, GHC shipped with a collection of packages under the name `extralibs`. Since GHC 6.12 the task of shipping an entire platform has been transferred to the Haskell Platform.

Recent progress

There has not been a release in the last 6 months. While the plan calls for major releases every 6 months this has not happened for a number of reasons. We took the decision not to base a major release on GHC-7.2.1 and no new release in the 7.2.x series is expected. We ran into some problems trying to prepare a release using GHC-7.0.4, however we may yet do a release using GHC-7.0.4.

Looking forward

Major releases are supposed to take place on a 6 month cycle. There will be a major release in Spring 2012 which will be based on the GHC-7.4.x series.

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

The platform steering committee will be proposing some modifications to the community review process for accepting new packages into the platform process with the aim of reducing the burden for package authors and keeping the review discussions productive. Though we will be making some modifications, we would still like to invite package authors to propose new packages. This can be initiated at any time. We also invite the rest of the community to take part in the review process on the libraries mailing list ⟨libraries@haskell.org⟩. The procedure involves writing a package proposal and discussing it on the mailing list with the aim of reaching a consensus. Details of the procedure are on the development wiki.

Further reading

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

3.2 The Glasgow Haskell Compiler

Report by: Simon Peyton Jones
Participants: many others

GHC 7.4.1 was released at the beginning of February, and has been by and large a successful release. Nevertheless the tickets keep pouring in, and a large collection of bug fixes [1] have been made since the 7.4.1 release. We plan to put out a 7.4.2 release candidate very soon (it may be out by the time you read this), followed shortly by the release.

We have a new member of the team! Please welcome Paolo Capriotti who is assuming some of the GHC maintenance duties for Well-Typed.

7.4.1 included a few major improvements. For more details on these, see the previous status report [2].

- Support for all declarations at the GHCi prompt
- Data type promotion and kind polymorphism [3]
- Improvements to Safe Haskell (safety is now inferred)
- Constraint Kinds [4]
- Profiling improvements: a major internal overhaul, and support for stack traces with `+RTS -xc`.
- Preliminary support for registerised ARM compilation (with full GHCi support being introduced in 7.4.2)

Here are the projects we're currently working on:

Kind polymorphism. Simon PJ has been working hard on completing the implementation of kind polymorphism and data type promotion [3]. This will appear for the first (supported) time in GHC 7.6; please do stress-test the HEAD.

Deferred type errors. Etienne Laurin suggested [16] that GHC could compile and run a program even though it contains type errors. After all, the bit you want to run might not contain the error, and it's sometimes annoying to have to fix every type error before you can run any code. It turned out that

there was a beautifully simple way to fit this idea into GHC's new constraint-based type inference engine, and we have now done so. It's all explained in "Equality proofs and deferred type errors" [17], and will be in GHC 7.6.

Holes in terms. Thijs Alkemade and Sean Leather have been working on another variant of deferred error messages, that would allow you to write a program that contains as-yet-unwritten sub-terms, or "holes" and have GHC report a fairly precise type for the hole. The idea is inspired by Agda's interactive programming environment, which has a facility of this kind. The more complicated the types get, the more useful this is! Details on their wiki page [18].

Type level literals. Iavor Diatchki has added type-level natural numbers (kind *Nat*) and strings (kind *Symbol*) to GHC. You can find lots of details on the wiki page [20]. At the moment there is no useful computation over the type-level naturals, but he is extending GHC's constraint solver with support for reasoning about type-expressions involving addition, multiplication, and exponentiation. This work is happening on branch `type-nats` in the repo, and we expect to have something working fairly soon.

Typechecker performance improvements. Most of the smarts of type inference are now located in the type constraint solver, described in our paper "Modular type inference with local assumptions: OutsideIn(X)" [19]. It works just fine for regular old ML-style programs, but was a bit slow for programs that make heavy use of type-level computation. Dimitrios has been working hard to improve its performance; we have carried out at least three major refactorings, deleted tons of code, and made it faster and more beautiful.

Windows x64 Support. The Industrial Haskell Group has funded work to implement 64bit Windows support in GHC. The port is now self-hosting and mostly complete, with just a number of bugs in the periphery to fix, and some logistics to work out. We expect a 64bit Windows installer to be included in the GHC 7.6 releases.

The new code generator. The glorious new code generator [6] has been an ongoing project for some time now. The basic idea is to replace the pass of the compiler that converts from STG to Cmm (our internal C– representation) with a more flexible framework consisting of two main passes: one that generates C– without explicit stack manipulation, and a second pass that makes the stack explicit. This will enable a host of improvements and optimisations in due course. The new code generator uses the Hoopl framework for code analysis and rewriting [7]. Earlier this year Simon M took over this project, and spent a lot of time optimising the existing framework and Hoopl itself. He also rewrote the stack allocator, and made a number of simplifications. The current state is that the new code generator produces code that is almost as good as the old one (and occasionally better), and is somewhat slower (roughly 15% slower compilation with -O). The goal is to further improve on this, and he's confident that we can generate better code in most cases than the old code generator. He hopes this can make it into 7.6.1, but no guarantees.

Changing the `+RTS -N` setting at runtime. Up until recently, the number of cores ("Capabilities" in GHC terminology) that GHC uses was fixed by the `+RTS -N` flag when you start the program. For instance, to use 2 cores, we pass the flag `+RTS -N2` to the Haskell program. GHC now has support for modifying this setting programmatically at runtime, both up and down, via the API *Control.Concurrent.setNumCapabilities*. So a parallel Haskell program can now set the number of cores to run on itself, without the user needing to pass `+RTS -N`. Another use for this feature is to drop back to using a single core during sequential sections of the program, which is likely to give better performance, especially on a loaded system. A threadscope diagram showing this in action is here: [5]. In the future we hope to use heuristics to dynamically adjust the number of cores in use according to system load or application demand, for example.

Profiling and stack traces. 7.4.1 has an overhauled profiling system, and in many cases gives better results than earlier versions. However, some details remain to be resolved around the precise semantics of cost-centre stacks. Also, Simon M hopes that it might be possible to provide stack traces of a kind without having to compile for profiling, perhaps in GHCi only.

Support for SSE primitives with LLVM back end. The simd git branch of GHC adds support for primitive 128-bit SIMD vector types and associated primops when using the LLVM back end, meaning this branch can now generate SSE instructions on x86 platforms. We hope this support will make it into 7.6.1. Experimental versions of the vector library [8] and DPH [9] provide higher-level interfaces to the new primitives. Initial benchmarks indicate that numerical code can benefit substantially.

Data Parallel Haskell. The vectorisation transformation underlying our implementation of nested data parallelism in GHC had a fundamental and long standing asymptotic complexity problem that we were finally able to resolve. Details are in a recent draft paper entitled "Work Efficient Higher-Order Vectorisation" [11]. The implementation described

in the paper is available in the DPH packages from Hackage (which needs to be used with GHC 7.4.1). The new implementation of the DPH libraries still needs to be optimised; hence, our next step will be to optimise constant factors.

In addition, we released Repa 3 [12], which uses type-indices to control array representations. This leads to more predictable performance. You can install Repa 3, which requires GHC 7.4.1, from Hackage. We are currently writing a paper describing the new design in detail.

Finally, we are about to release (it may be out by the time you read this) a stable, end-user ready version of the Repa-like array library Accelerate for GPU computing on Hackage. It integrates with Repa, so you can mix GPU and CPU multicore computing, and via the new `meta-par` package you can share workload between CPUs and GPUs [13]. This new version 0.12 is already available on GitHub [14]. You need a CUDA-capable NVIDIA GPU to use it.

Lightweight concurrency substrate. During his internship at MSR Cambridge, Sivaramakrishnan Krishnamoorthy Chandrasekaran (aka "KC") has been working on replacing the RTS scheduler with some APIs that enable the scheduler to be implemented in Haskell. The aim is to not just move the scheduler into Haskell, but also enable user-defined schedulers to coexist, which will ultimately enable much greater control over scheduling behaviour. This follows on from previous work [15] with Peng Li and Andrew Tolmach, but this time we are taking a slightly different approach that has a couple of important benefits.

Firstly, KC found a way to enable concurrency abstractions to be defined without depending on a particular scheduler. This means for example that we can provide MVars that work with any user-defined scheduler, rather than needing one MVar implementation per scheduler. Secondly, we found ways to coexist with some of the existing RTS machinery for handling blackholes and asynchronous exceptions in particular, which means that these facilities will continue to work as before (with the same performance), and writers of user-defined schedulers do not need to worry about them. Furthermore this significantly lowers the barrier for writing a new scheduler.

This is all still very much experimental, and it is not clear whether it will ever be in GHC proper. It depends on whether we can achieve good enough performance, amongst other things. All we can say for now is that the approach is promising. You can find KC's work on the ghc-lwc branch of the git repo.

Full support for GHCi on ARM. Thanks to Ben Gamari, we now have support for ARM in the GHCi linker [21]. This will be shipped in 7.4.2 (it wasn't in 7.4.1).

Bibliography

- [1] Towards Haskell in the Cloud, Jeff Epstein, Andrew P. Black, and Simon Peyton Jones, Haskell Symposium 2011, http://research.microsoft.com/~simonpj/papers/parallel/remote.pdf
- [1] http://hackage.haskell.org/trac/ghc/query?status=closed&order=priority&col=id&col=summary&col=status&col=owner&col=type&col=priority&col=component&milestone=7.4.2&resolution=fixed
- [2] http://hackage.haskell.org/trac/ghc/wiki/Status/Oct11
- [3] http://www.haskell.org/ghc/docs/7.4.1/html/users_guide/kind-polymorphism-and-promotion.html#kind-polymorphism
- [4] http://www.haskell.org/ghc/docs/7.4.1/html/users_guide/constraint-kind.html
- [5] https://plus.google.com/107890464054636586545/posts/GsfcJfdkEYL
- [6] http://hackage.haskell.org/trac/ghc/wiki/Commentary/Compiler/NewCodeGen
- [7] http://www.cs.tufts.edu/~nr/pubs/dfopt-abstract.html
- [8] http://ghc-simd.blogspot.co.uk/2012/03/simd-support-for-vector-library.html
- [9] http://ghc-simd.blogspot.co.uk/2012/04/adding-simd-support-to-data-parallel.html
- [11] http://www.cse.unsw.edu.au/~chak/papers/LCKLP12.html
- [12] http://repa.ouroborus.net/
- [13] http://parfunk.blogspot.com.au/2012/05/how-to-write-hybrid-cpugpu-programs.html
- [14] https://github.com/AccelerateHS/accelerate
- [15] http://community.haskell.org/~simonmar/papers/conc-substrate.pdf
- [16] Deferring type errors to runtime. http://hackage.haskell.org/trac/ghc/wiki/DeferErrorsToRuntime
- [17] Equality proofs and deferred type errors. http://research.microsoft.com/en-us/um/people/simonpj/papers/ext-f/
- [18] Holes in GHC. http://hackage.haskell.org/trac/ghc/wiki/Holes
- [19] Modular type inference with local assumptions: OutsideIn(X). http://www.haskell.org/haskellwiki/Simonpj/Talk:OutsideIn

- [20] Type level literals. http://hackage.haskell.org/trac/ghc/wiki/TypeNats/Basics
- [21] ARM linker support. http://hackage.haskell.org/trac/ghc/changeset/27302c9094909e04eb73f200d52d5e9370c34a8a

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 (→ 5.3.1).

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 350 ports of many popular Cabal packages. The updates committed to this repository are continuously integrated to the official ports tree on a regular basis. Though the FreeBSD Ports Collection already has many popular and important Haskell software: GHC 7.0.4, Haskell Platform 2011.4.0.0, Gtk2Hs, wxHaskell, XMonad, Pandoc, Gitit, Yesod, Happstack, and Snap — that have been incorporated into the recently published FreeBSD 8.3-RELEASE.

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

Further reading

http://wiki.FreeBSD.org/Haskell

3.4.2 Debian Haskell Group

Report by:	Joachim Breitner
Status:	working

The Debian Haskell Group aims to provide an optimal Haskell experience to users of the Debian GNU/Linux distribution and derived distributions such as Ubuntu. We try to follow the Haskell Platform versions for the core package and package a wide range of other useful libraries and programs. At the time of writing, we maintain 500 source packages.

A system of virtual package names and dependencies, based on the ABI hashes, guarantees that a system upgrade will leave all installed libraries usable. Most

libraries are also optionally available with profiling enabled and the documentation packages register with the system-wide index.

The stable Debian release ("squeeze") provides the Haskell Platform 2010.1.0.0 and GHC 6.12, Debian testing ("wheezy") contains the Platform version 2011.4.0.0 with GHC 7.0.4 and in unstable we are currently ahead of the Platform and ship GHC 7.4.1. We plan to get GHC 7.4.2 and the Platform version 2012.2.0.0 into wheezy in time before the stable release, expected this year.

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

Further reading

http://wiki.debian.org/Haskell

3.4.3 Haskell in Gentoo Linux

Report by:	Sergei Trofimovich

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

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

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

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

There is also an overlay which contains almost 800 extra unofficial and testing packages. Thanks to the Haskell developers using Cabal and Hackage (→ 6.6.1), 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, Ben Boeckel, Michel Salim, Shakthi Kannan, and others
Status:	ongoing

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

Fedora 17 is shipping in May with ghc-7.0.4 and haskell-platform-2011.4.0.0, and version updates to many of the packages. This also includes Fedora 17 Secondary architectures: ppc, ppc64, and the exciting new armv5tel and armv7hp builds (ghc has also been built for Fedora 17 s390 and s390x for the first time). 30 new packages have been added since the release of Fedora 16, including aeson, conduit, hakyll, lifted-base, snap-core, warp, etc.

On the packaging side, for Fedora 16 profiling subpackages were merged into the development subpackages to reduce installation overhead. For Fedora 17 the packaging macros have been simplified and made closer to generic Fedora packaging.

At the time of writing there are now 165 Haskell source packages in Fedora. The Fedora package version numbers listed on the Hackage website now refer to the latest branched version of Fedora (currently 17).

Fedora 18 development work has already started and we have already updated to ghc-7.4.1 and continue work on packaging including web frameworks.

Feedback from users and packaging contributions to Fedora Haskell are always welcome: please join us on #fedora-haskell on Freenode IRC and our new low-traffic mailing-list.

Further reading

- Homepage: http://fedoraproject.org/wiki/SIGs/Haskell
- New 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/git/?p=haskell-sig.git;a=blob;f=packages/diffs/f16-f17.diff

3.5 Fibon Benchmark Tools & Suite

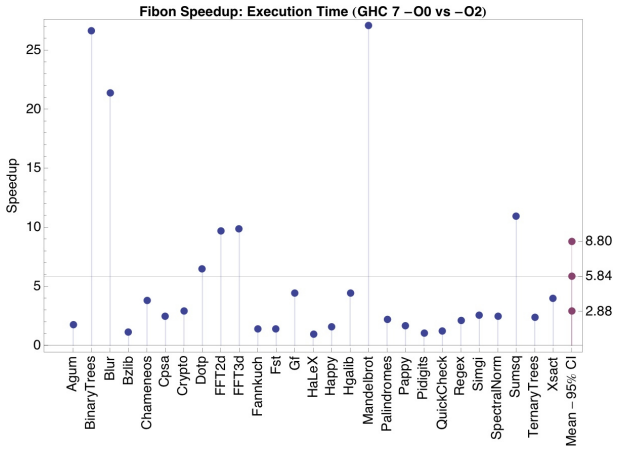
Report by:	David M. Peixotto
Status:	stable

Fibon is a set of tools for running and analyzing benchmark programs in Haskell. It contains an optional set of benchmarks from various sources including several programs from the Hackage repository.

The Fibon benchmark tools draw inspiration from both the venerable nofib Haskell benchmark suite and the industry standard SPEC benchmark suite. The tools automate the tedious parts of benchmarking: building the benchmark in a sand-boxed directory, running the benchmark multiple times, verifying correctness, collecting statistics, and summarizing results.

Benchmarks are built using the standard `cabal` tool. Any program that has been cabalized can be added as benchmark simply by specifying some meta-information about the program inputs and expected outputs. Fibon will automatically collect execution times for benchmarks and can optionally read the statistics output by the GHC runtime. The program outputs are checked to ensure correct results making Fibon a good option for testing the safety and performance of program optimizations. The Fibon tools are not tied to any one benchmark suite. As long as the correct meta-information has been supplied, the tools will work with any set of programs.

As a real life example of a complete benchmark suite, Fibon comes with its own set of benchmarks for testing the effectiveness of compiler optimizations in GHC. The benchmark programs come from Hackage, the Computer Language Shootout, Data Parallel Haskell, and Repa. The benchmarks were selected to have minimal external dependencies so they could be easily used with a version of GHC compiled from the latest sources. The following figure shows the performance improvement of GHC's optimizations on the Fibon benchmark suite.



This year, the Fibon benchmark suite has been updated to include a `Train` problem size that can be used for feedback directed optimization work. The `Ref` problem size has been increased so that the running time of a benchmark program is comparable to the running time when using the ref size of the SPEC benchmarks. With this update a single benchmark will typically take $10 - 30$ minutes to run depending on the power of the computer hardware. See the README file for more information on benchmark size and configuring the benchmarks to finish in an acceptable amount of time.

The Fibon tools and benchmark suite are ready for public consumption. They can be found on github at the url indicated below. People are invited to use the included benchmark suite or just use the tools and build a suite of their own creation. Any improvements to the tools or additional benchmarks are most welcome. Benchmarks have been used to tell lies about performance for many years, so join in the fun and keep on fibbing with Fibon.

Further reading

- https://github.com/dmpots/fibon
- https://github.com/dmpots/fibon-benchmarks
- https://github.com/dmpots/fibon-config

4 Related Languages

4.1 Agda

Report by:	Nils Anders Danielsson
Participants:	Ulf Norell, Andreas Abel, and many others
Status:	actively developed

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

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

The next version of Agda is under development. The most interesting changes to the language may be the addition of pattern synonyms, contributed by Stevan Andjelkovic and Adam Gundry, and modifications of the constraint solver, implemented by Andreas Abel. Other work has targeted the Emacs mode. Peter Divianszky has removed the prior dependency on GHCi and haskell-mode, and Guilhem Moulin and myself have made the Emacs mode more interactive: type-checking no longer blocks Emacs, and the expression that is currently being type-checked is highlighted.

Further reading

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

4.2 MiniAgda

Report by:	Andreas Abel
Status:	experimental

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

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

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

Further reading

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

4.3 Disciple

Report by:	Ben Lippmeier
Participants:	Erik de Castro Lopo
Status:	experimental, active development

Disciple is a dialect of Haskell that uses strict evaluation as the default and supports destructive update of arbitrary data. Many Haskell programs are also Disciple programs, or will run with minor changes. In addition, Disciple includes region, effect, and closure typing, and this extra information provides a handle on the operational behaviour of code that is not available in other languages. Our target applications are the ones that you always find yourself writing C programs for, because existing functional languages are too slow, use too much memory, or do not let you update the data that you need to.

Our compiler (DDC) is still in the "research prototype" stage, meaning that it will compile programs if you are nice to it, but expect compiler panics and missing features. You will get panics due to ungraceful handling of errors in the source code, but valid programs should compile ok. The test suite includes a few thousand-line graphical demos, like a ray-tracer and an n-body collision simulation, so it is definitely hackable.

Over the last six months we continued working towards mechanising the metatheory of the DDC core language in Coq. We've finished Progress and Preservation for System-F2 with mutable algebraic data, and are now looking into proving contextual equivalence of rewrites in the presence of effects. Based on this experience, we've also started on an interpreter for a cleaned up version of the DDC core language. We've taken the advice of previous paper reviewers and removed dependent kinds, moving witness expressions down to level 0 next to value expressions. In the resulting language, types classify both witness and value expressions, and kinds classify types. We're also removing more-than constraints on effect and closure variables, along with dangerous type variables (which never really worked).

All over, it's being pruned back to the parts we understand properly, and the removal of dependent kinds will make mechanising the metatheory easier. Writing an interpreter for the core language also gets us a parser for it, which we will need for performing cross module inlining in the compiler proper.

Further reading

http://disciple.ouroborus.net

5 Haskell and ...

5.1 Haskell and Parallelism

5.1.1 Eden

Report by:	Rita Loogen
Participants:	**in Madrid:** Yolanda Ortega-Mallén, Mercedes Hidalgo, Lidia Sánchez-Gil, Fernando Rubio, Alberto de la Encina, **in Marburg:** Mischa Dieterle, Thomas Horstmeyer, 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 #05. 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.

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, Lecture Notes, CEFP Summer School, Budapest, Hungary, June 2011, Springer LNCS 2741, 2012 (to appear).
(see also: http://www.mathematik.uni-marburg.de/~eden/?content=cefp)

Implementation

A new release of the Eden compiler based on GHC 7.4 will soon be available on our web pages, see http://www.mathematik.uni-marburg.de/~eden, and via Hackage. It will include a shared memory mode which does not depend on a middleware like MPI but which nevertheless uses multiple independent heaps (in contrast to GHC's threaded runtime system) connected by Eden's parallel runtime system. An Eden variant of GHC-7.4 and the Eden libraries are already available via git repositories at http://james.mathematik.uni-marburg.de:8080.

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 has been written in Haskell and is freely available on the Eden web pages.

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

Recent and Forthcoming Publications

- Oleg Lobachev: *Implementation and Evaluation of Algorithmic Skeletons: Parallelisation of Computer Algebra Algorithms*, Ph.D. thesis, Philipps-Universität Marburg, Germany, October 2011.
- Rita Loogen: *Eden - Parallel Functional Programming in Haskell*, Lecture Notes, CEFP Summer School, Budapest, Hungary, June 2011, Springer LNCS 7241, 65 pages, 2012 (to appear).
- Jost Berthold, Andrzej Filinski, Fritz Henglein, Ken Friis Larsen, Mogens Steffensen, and Brian Vinter: *Functional High Performance Financial IT — The HIPERFIT Research Center in Copenhagen*, Trends in Functional Programming (TFP'11) — Revised Selected Papers, Springer LNCS (to appear).

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:	Eric Kow
Participants:	Duncan Coutts, Andres Löh, Nicolas Wu, Mikolaj Konarski, Edsko de Vries
Status:	active

Microsoft Research is funding a 2-year project to promote the real-world use of parallel Haskell. The project started in November 2010, with four industrial partners, and consulting and engineering support from Well-Typed (→ 8.1). Each organisation is working on its own particular project making use of parallel Haskell. The overall goal is 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.

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

The two main areas of focus in the project recently have been ThreadScope and Cloud Haskell.

ThreadScope The latest release of ThreadScope (version 0.2.1) provides new visualisations that allow the user to observe the creation and conversion of sparks into actual work. These visualisations are aimed at giving users of ThreadScope more insight into the performance of their programs, not just what programs are doing performance-wise, but why.

Much of the ThreadScope work leading up to this release consists in backend investments, improvements to the ghc-events package (a new state machine representation of the meaning of events) and the GHC runtime system (adding a new startup wall-clock time and Haskell thread labels to the event log). These changes will enable more useful improvements to ThreadScope in the future.

The release is also accompanied by a new tutorial on the Haskell wiki, the ThreadScope Tour. The tours provides a series of self-contained miniature walkthroughs focusing on various aspects of ThreadScope usage, for example, observing the need to consolidate sequential evaluation in order to make ThreadScope output easier to interpret.

Cloud Haskell We have been working on Cloud Haskell for distributed parallelism. In particular, we are developing a new implementation that is intended to be robust, flexible and have good performance. The resulting "distributed-process" package will build off an internal design which includes a swappable network transport layer. As we flesh out this implementation, we are also working on further developing and validating the new design. These ongoing efforts are visible from the GitHub page listed below.

Summary

- ThreadScope (0.2.1), with further improvements to spark profiling
- ghc-events (0.4.0.0) with state machine representation of events
- distributed-process, ongoing work towards a new implementation of Cloud Haskell with a swappable transport layer, https://github.com/haskell-distributed/distributed-process
- ThreadScope Tour, a guided tour of ThreadScope, http://www.haskell.org/haskellwiki/ThreadScope_Tour

5.1.4 Static Verification of Transactions in STM Haskell

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

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

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

Contact

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

5.2 Haskell and the Web

5.2.1 WAI

Report by:	Greg Weber
Status:	stable

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

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

WAI is also a platform for re-using code between web applications and web frameworks through WAI middleware and WAI applications. WAI middleware can inspect and transform a request, for example by automatically gzipping a response or logging a request.

By targeting WAI, every web framework can share WAI code instead of wasting effort re-implementing the same functionality. There are also some new web frameworks that take a completely different approach to web development that use WAI, such as webwire (FRP) and dingo (GUI). Since the last HCAR, another web framework called Scotty was released. WAI applications can send a response themselves. For example, wai-app-static is used by Yesod to serve static files. However, one does not need to use a web framework, but can simply build a web application using the WAI interface alone. The Hoogle web service targets WAI directly.

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

Further reading

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

5.2.2 Warp

Report by:	Greg Weber

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

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

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

"Warp: A Haskell Web Server" by Michael Snoyman was published in the May/June 2011 issue of IEEE Internet Computing:

- Issue page: http://www.computer.org/portal/web/csdl/abs/mags/ic/2011/03/mic201103toc.htm
- PDF: http://steve.vinoski.net/pdf/IC-Warp_a_Haskell_Web_Server.pdf

5.2.3 Holumbus Search Engine Framework

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

Description

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

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

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

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

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

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

Features

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

Current Work

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

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

The Happstack project is focused on bringing the relentless, uncompromised power and beauty of Haskell to a web framework. We aim to leverage the unique characteristics of Haskell to create a highly-scalable, robust, and expressive web framework.

While Happstack is over 7 years old, it is still undergoing active development and new innovation. It is used in a number of commercial projects as well as the new Hackage 2 server.

At the core of Happstack is the happstack-server package which provides a fast, powerful, and easy to use HTTP server with built-in support for templating (via blaze-html), request routing, form-decoding, cookies, file-uploads, etc. happstack-server is all you need to create a simple website.

Happstack can also be extended using a wide range of libraries which include support for alternative HTML templating systems, javascript templating and generation, type-safe URLs, type-safe form generation and validation, RAM-cloud database persistence, OpenId authentication, and more.

Future plans

Upcoming innovations we will be exploring in Happstack include:

- more powerful and flexible 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
- better support for reusable web components (such as components for authentication, threaded comments, etc)
- fundamental architecture changes to the HTTP backend which will allow for greater scalability and greater assurances of correctness

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
- http://happstack.blogspot.com/

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 providing basic web features and CGI (mailman and contents search).

Mighttpd version 1 was implemented with two libraries *c10k* and *webserver*. Since GHC 6 uses select(), more than 1,024 connections cannot be handled at the same time. The *c10k* library gets over this barrier with the pre-fork technique. The *webserver* library provides HTTP transfer and file/CGI handling.

Mighttpd 2 stops using the *c10k* library because GHC 7 starts using epoll()/kqueue(). The file/CGI handling part of the *webserver* library is reimplemented as a web application on the *wai* library (→ 5.2.1). For HTTP transfer, Mighttpd 2 links the *warp* library (→ 5.2.2) which can send a file in zero copy manner thank to sendfile().

The performance of Mighttpd 2 is now comparable to highly tuned web servers written in C Please read "The Monad.Reader" Issue 19 for more information.

Mighttpd 2 is now based on Conduit version 0.4 and provides the functionality of reverse proxy. You can install Mighttpd 2 (*mighttpd2*) from HackageDB.

Further reading

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

5.2.6 Yesod

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

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

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

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

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

- declarative routing with type-safe urls — say goodbye to broken links
- no XSS attacks — form submissions are automatically sanitized
- database safety through the Persistent library (→ 7.7.2) — no SQL injection and queries are always valid
- valid template variables with proper template insertion — variables are known at compile time and treated differently according to their type using the shakesperean templating system.

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

MVC stands for model-view-controller. The preferred library for models is Persistent (→ 7.7.2). View can be handled by the Shakespeare family of compile-time template languages. This includes Hamlet, which takes the tedium out of HTML. Both of these libraries are optional, and you can use any Haskell alternative. Controllers are invoked through declarative routing. Their return type shows which response types are allowed for the request.

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

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

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

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

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

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

Further reading

http://www.yesodweb.com/

5.2.7 Snap Framework

Report by:	Doug Beardsley
Participants:	Gregory Collins, Shu-yu Guo, James Sanders, Carl Howells, Shane O'Brien, Ozgun Ataman, Chris Smith, Jurrien Stutterheim, and others
Status:	active development

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

The Snap Framework has seen two major releases (0.7 and 0.8) since the last HCAR. Some of the major features added are better awareness of proxy servers and address translation, more powerful timeout handling, more control over buffering semantics, improvements to the test infrastructure, and a number of other bug fixes and minor improvements.

We are starting to see more high level functionality developed by third parties being made available as snaplets. A complete list of the third-party snaplets we are aware of can be found in the snaplet directory page on our website. So far this includes seven different snaplets providing support for various data stores, support for different build environments, ReCAPTCHA support, and a snaplet providing functionality similar to "rake tasks" from Ruby on Rails.

Further reading

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

5.2.8 Ivy-web

Report by:	James Deng
Status:	experimental

Ivy-web is a lightweight web framework, with type safe routes, based on invertible-syntax, and i18n support, influenced by Django, Snap, and Yesod.

The features of this web framework:

- Type safe routes, specify url-handler mapping in one place. For example, we want a url mapping for blog as "/blog/year-month-day" to Handler Int Int Int, where year, month and day are integers. We can declare as follows:

```
data Blog = Blog Int Int Int
          deriving (Show, Eq, Typeable)
$ (defineIsomorphisms '' Blog)
instance Handler Blog where
  get b@(Blog y m d) _ = do
    t ← liftIO getClockTime
    return $ responseHtml $ trans' "blog"
           ++ show b ++ show t
  rBlog = blog <$> text "/blog/" *>
                   int <-> int <-> int
```

  We can reverse this mapping from handler value automatically, thus do not need to construct url string manually in code, avoiding url errors.

```
ghci> url (Blog 2011 9 19) ,
      == "/blog/2009-9-19",
```

- Simple yet elegant handler via type class.

```
class Handler a where
  get, post, put, delete, handle :: a → Application
  handle a req = case requestMethod req of
    m | m ≡ methodGet → get a req
      | m ≡ methodPost → post a req
      | m ≡ methodPut → put a req
      | m ≡ methodDelete → delete a req
    otherwise → unimplemented req
```

- Flexible template system, utilize exsisting libraries such as Blaze-Html and Hastache.
- Easy i18n — Wraps around i18n library.
- TODO: Auth system — Port from snap-auth.
- TODO: Modular app system like Django — The current route system support modular routes very well. Need works in modular config and data files like static template files.
- TODO: Persistent library — Improving the DSH library is my current preference.

The principle of this library is KISS, and "don't reinvent the wheel" by reusing existing state-of-the-art libraries.

For the example code listed above, please refer to https://github.com/lilac/ivy-example/

Recent developments

I have ported ivy-web from wai to snap-server backend, and also wrote a sample project correspond to the starter project of snap. When everything is fine and I am free, I will upload the code and bump the version to 0.2.

Further reading

- https://github.com/lilac/ivy-web/
- http://hackage.haskell.org/package/ivy-web

5.2.9 rss2irc

Report by:	Simon Michael
Status:	beta

rss2irc is an IRC bot that polls a single RSS or Atom feed and announces new items to an IRC channel, with options for customizing output and behavior. It aims to be an easy to use, dependable bot that does its job and creates no problems.

rss2irc was published in 2008 by Don Stewart. Simon Michael took over maintainership in 2009, with the goal of making a robust low-maintenance bot to stimulate development in various free/open-source software communities. It is currently used for several full-time bots including:

- hackagebot — announces new hackage releases in #haskell
- hledgerbot — announces hledger commits in #ledger
- zwikicommitbot — announces Zwiki commits in #zwiki
- squeaksobot — announces Squeak and Smalltalk-related Stack Overflow questions in #squeak
- squeakquorabot — announces Squeak/Smalltalk-related Quora questions in #squeak
- etoystrackerbot — announces new Etoys bugs in #etoys
- etoysupdatesbot — announces Etoys commits in #etoys
- planetzopebot — announces new planet.zope.org posts in #zope

The project is available under BSD license from its home page at http://hackage.haskell.org/package/rss2irc.

Since last report there has been a great deal of cleanup and enhancement, but no new release on hackage yet due to an xml-related memory leak.

Further reading

http://hackage.haskell.org/package/rss2irc

5.3 Haskell and Compiler Writing

5.3.1 UUAG

Report by:	Arie Middelkoop
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 0.9.39 (October 2011), is extensively tested, and is available on Hackage. Recently, we improved the Cabal support and ensured compatibility with GHC 7.

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

First-class AGs We provide a translation from UUAG to AspectAG (→ 5.3.2). 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.

Multi-core evaluation Our algorithm for ordered AGs identifies statically which subcomputations of children of a production are independent and suitable for parallel evaluation. Together with the strict evaluation as mentioned above, which is important when evaluating in parallel, the generated code can automatically exploit multi-core CPUs. We are currently evaluating the effectiveness of this approach.

Stepwise evaluation In the recent past we worked on a stepwise evaluation scheme for AGs. Using this scheme, the evaluation of a node may yield user-defined progress reports, and the evaluation to the next report is considered to be an evaluation step. By asking nodes to yield reports, we can encode the parallel exploration of trees and encode breadth-first search strategies.

We are currently also running a Ph.D. project that investigates incremental evaluation.

Further reading

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

5.3.2 AspectAG

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

AspectAG is a library of strongly typed Attribute Grammars implemented using type-level programming.

Introduction

Attribute Grammars (AGs), a general-purpose formalism for describing recursive computations over data types, avoid the trade-off which arises when building software incrementally: should it be easy to add new data types and data type alternatives or to add new operations on existing data types? However, AGs are usually implemented as a pre-processor, leaving e.g. type checking to later processing phases and making interactive development, proper error reporting and debugging difficult. Embedding AG into Haskell as a combinator library solves these problems. Previous attempts at embedding AGs as a domain-specific

language were based on extensible records and thus exploiting Haskell's type system to check the well-formedness of the AG, but fell short in compactness and the possibility to abstract over oft occurring AG patterns. Other attempts used a very generic mapping for which the AG well-formedness could not be statically checked. We present a typed embedding of AG in Haskell satisfying all these requirements. The key lies in using HList-like typed heterogeneous collections (extensible polymorphic records) and expressing AG well-formedness conditions as type-level predicates (i.e., typeclass constraints). By further type-level programming we can also express common programming patterns, corresponding to the typical use cases of monads such as Reader, Writer, and State. The paper presents a realistic example of type-class-based type-level programming in Haskell.

We have included support for local and higher-order attributes. Furthermore, a translation from UUAG to AspectAG is added to UUAGC as an experimental feature.

Current Status

We have recently added a combinator *agMacro* to provide support for "attribute grammars macros"; a mechanism that makes it easy to define attribute computation in terms of already existing attribute computation.

Background

The approach taken in AspectAG was proposed by Marcos Viera, Doaitse Swierstra, and Wouter Swierstra in the ICFP 2009 paper "Attribute Grammars Fly First-Class: How to do aspect oriented programming in Haskell".

The Attribute Grammar Macros combinator is described in a technical report: UU-CS-2011-028.

Further reading

http://www.cs.uu.nl/wiki/bin/view/Center/AspectAG

5.3.3 LQPL — A Quantum Programming Language Compiler and Emulator

Report by:	Brett G. Giles
Participants:	Dr. J.R.B. Cockett
Status:	v 0.9.0 experimental releasing in May 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 modules. These consist of:

- The compiler — available at the command line and via a TCP/IP interface.
- The emulator — available as a server via a TCP/IP interface.
- The front end — with version 0.9, the front end is written as a Java/Swing application, which connects to both the compiler and the emulator via TCP/IP. Further front ends are being contemplated.

During the modification to create these separate modules, Hspec was used to verify the interfaces worked as designed.

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 is a preparatory step for improving the performance of the emulator and adding optimization features to the language.

Further reading

http://pll.cpsc.ucalgary.ca/lqpl/index.html

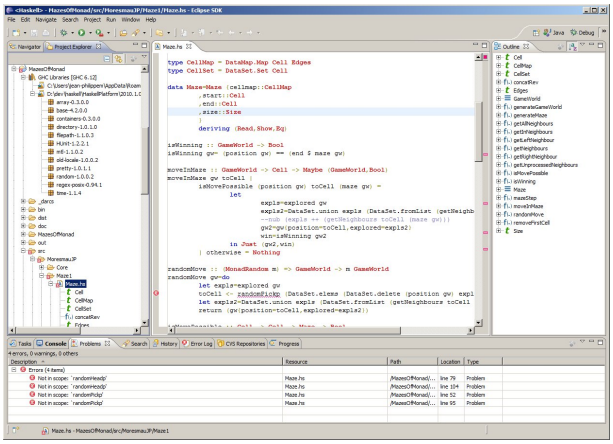
6 Development Tools

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, and others
Status:	stable, maintained, and actively developed

EclipseFP is a set of Eclipse plugins to allow working on Haskell code projects. It features Cabal integration (.cabal file editor, uses Cabal settings for compilation, allows the user to install Cabal packages from within the IDE), and GHC integration. Compilation is done via the GHC API, syntax coloring uses the GHC Lexer. Other standard Eclipse features like code outline, folding, and quick fixes for common errors are also provided. HLint suggestions can be applied in one click. EclipseFP also allows launching GHCi sessions on any module including extensive debugging facilities. It uses BuildWrapper to bridge between the Java code for Eclipse and the Haskell APIs. It also provides a full package and module browser to navigate the Haskell packages installed on your system, integrated with Hackage. The source code is fully open source (Eclipse License) on github and anyone can contribute. Current version is 2.2.4, released in March 2012 and supporting GHC 7.0 and above, and more versions with additional features are planned and actively worked on. Feedback on what is needed is welcome! The website has information on downloading binary releases and getting a copy of the source code. Support and bug tracking is handled through Sourceforge forums.



Further reading

http://eclipsefp.github.com/

6.1.2 ghc-mod — Happy Haskell Programming

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

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

Emacs front-end provides the following features:

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

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

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

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

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

There are two Vim plugins:

- ghcmod-vim
- syntastic

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

A new major version of Heat has appeared, which

- works on top of GHCi instead of Hugs,
- supports automatic QuickCheck property testing,
- uses a simple model of updating Haskell files in place,

- is distributed as a single jar file.

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 portable, small and works on top of a Haskell interpreter.

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/

6.1.4 HaRe — The Haskell Refactorer

Report by:	Simon Thompson
Participants:	Huiqing Li, Chris Brown, Claus Reinke

See: http://www.haskell.org/communities/05-2011/html/report.html#sect5.1.5.

6.2 Documentation

6.2.1 Haddock

Report by:	David Waern
Status:	experimental, maintained

Haddock is a widely used documentation-generation tool for Haskell library code. Haddock generates documentation by parsing and typechecking Haskell source code directly and including documentation supplied by the programmer in the form of specially-formatted comments in the source code itself. Haddock has direct support in Cabal (→ 6.6.1), and is used to generate the documentation for the hierarchical libraries that come with GHC, Hugs, and nhc98 (http://www.haskell.org/ghc/docs/latest/html/libraries) as well as the documentation on Hackage.

The latest release is version 2.9.4, released October 3 2011.

Recent changes:

- Support for GHC 7.2 and Alex 3.x
- New –qual flag for qualification of names
- Print doc coverage information to stdout
- Speed up generation of index
- Various bug fixes

Future plans

- Although Haddock understands many GHC language extensions, we would like it to understand all of them. Currently there are some constructs you cannot comment, like GADTs and associated type synonyms.
- Error messages is an area with room for improvement. We would like Haddock to include accurate line numbers in markup syntax errors.
- On the HTML rendering side we want to make more use of Javascript in order to make the viewing experience better. The frames-mode could be improved this way, for example.
- Finally, the long term plan is to split Haddock into one program that creates data from sources, and separate backend programs that use that data via the Haddock API. This will scale better, not requiring adding new backends to Haddock for every tool that needs its own format.

Further reading

- Haddock's homepage: http://www.haskell.org/haddock/
- Haddock's developer Wiki and Trac: http://trac.haskell.org/haddock
- Haddock's mailing list: haddock@projects.haskell.org

6.2.2 lhs2TEX

Report by:	Andres Löh
Status:	stable, maintained

This tool by Ralf Hinze and Andres Löh is a preprocessor that transforms literate Haskell or Agda code into LaTeX documents. The output is highly customizable by means of formatting directives that are interpreted by lhs2TEX. Other directives allow the selective inclusion of program fragments, so that multiple versions of a program and/or document can be produced from a common source. The input is parsed using a liberal parser that can interpret many languages with a Haskell-like syntax.

The program is stable and can take on large documents.

The current version is 1.17, so there has not been a new release since the last report. 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

6.3 Testing and Analysis

6.3.1 shelltestrunner

Report by:	Simon Michael
Status:	stable

shelltestrunner was first released in 2009, inspired by the test suite in John Wiegley's ledger project. It is a command-line tool for doing repeatable functional testing of command-line programs or shell commands. It reads simple declarative tests specifying a command, some input, and the expected output, error output and exit status. Tests can be run selectively, in parallel, with a timeout, in color, and/or with differences highlighted.

In the last six months, shelltestrunner has had three releases (1.0, 1.1, 1.2) and acquired a home page. Projects using it include hledger, yesod, berp, and eddie. shelltestrunner is free software released under GPLv3+ from Hackage or http://joyful.com/shelltestrunner.

Further reading

http://joyful.com/repos/shelltestrunner

6.3.2 hp2any

Report by:	Patai Gergely
Status:	experimental

This project was born during the 2009 Google Summer of Code under the name "Improving space profiling experience". The name `hp2any` covers a set of tools and libraries to deal with heap profiles of Haskell programs. At the present moment, the project consists of three packages:

- `hp2any-core`: a library offering functions to read heap profiles during and after run, and to perform queries on them.
- `hp2any-graph`: an OpenGL-based live grapher that can show the memory usage of local and remote processes (the latter using a relay server included in the package), and a library exposing the graphing functionality to other applications.
- `hp2any-manager`: a GTK application that can display graphs of several heap profiles from earlier runs.

The project also aims at replacing `hp2ps` by reimplementing it in Haskell and possibly adding new output formats. The manager application shall be extended to display and compare the graphs in more ways, to export them in other formats and also to support live profiling right away instead of delegating that task to `hp2any-graph`.

Recently, the `hp2any` project joined forces with `hp2pretty`, which resulted in increased performance in the core library.

Further reading

- http://www.haskell.org/haskellwiki/Hp2any
- http://code.google.com/p/hp2any/
- http://gitorious.org/hp2pretty

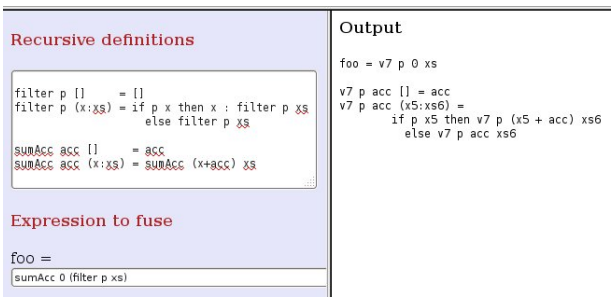
6.4 Optimization

6.4.1 HFusion

Report by:	Facundo Dominguez
Participants:	Alberto Pardo
Status:	experimental

HFusion is an experimental tool for optimizing Haskell programs. The tool performs source to source transformations by the application of a program transformation technique called *fusion*. The aim of fusion is to reduce memory management effort by eliminating the intermediate data structures produced in function compositions. It is based on an algebraic approach where functions are internally represented in terms of a recursive program scheme known as *hylomorphism*.

We offer a web interface to test the technique on user-supplied recursive definitions and HFusion is also available as a library on Hackage. The last improvement to HFusion has been to accept as input an expression containing any number of compositions, returning the expression which results from applying fusion to all of them. Compositions which cannot be handled by HFusion are left unmodified.



In its current state, HFusion is able to fuse compositions of general recursive functions, including primitive recursive functions (like dropWhile or insertions in binary search trees), functions that make recursion over

multiple arguments like zip, zipWith or equality predicates, mutually recursive functions, and (with some limitations) functions with accumulators like foldl. In general, HFusion is able to eliminate intermediate data structures of regular data types (sum-of-product types plus different forms of generalized trees).

Further reading

- HFusion publications: http://www.fing.edu.uy/inco/proyectos/fusion
- HFusion web interface: http://www.fing.edu.uy/inco/proyectos/fusion/tool
- HFusion on Hackage: http://hackage.haskell.org/package/hfusion

6.4.2 Optimizing Generic Functions

Report by:	José Pedro Magalhães
Participants:	Johan Jeuring, Andres Löh
Status:	actively developed

See: http://www.haskell.org/communities/11-2010/html/report.html#sect8.5.4.

6.5 Code Management

6.5.1 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 2.5.2, was in March 2011. We are very close to releasing Darcs 2.8 (the second release candidate is out). Some key changes include support for GHC 7, a faster and more readable `darcs annotate`, a `darcs obliterate -O` which can be used to conveniently "stash" patches, hunk editing for the `darcs revert` command.

Over the longer term, Darcs will emphasise three development priorities

1. Improving code quality: this ranges from surface-level improvements such as switching to a uninform coding style, to deeper refactors and a move towards a more principled separation of Darcs subsystems.
2. Supporting Darcs hosting and GUIs: we aim to provide library code that makes it easier to write hosting sites such as Darcsden and Patch-Tag, or graphical interfaces to Darcs. This work may potentially involve writing prototype hosting code to test our library.
3. Developing the Darcs 3 theory of patches: we aim specifically to address the conflict-resolution issues that Darcs suffers from.

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

6.5.2 DarcsWatch

Report by:	Joachim Breitner
Status:	working

DarcsWatch is a tool to track the state of Darcs (→ 6.5.1) patches that have been submitted to some project, usually by using the `darcs send` command. It allows both submitters and project maintainers to get an overview of patches that have been submitted but not yet applied.

DarcsWatch continues to be used by the xmonad project (→ 7.8.2), the Darcs project itself, and a few developers. At the time of writing, it was tracking 39 repositories and 4288 patches submitted by 234 users.

Further reading

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

6.5.3 darcsden

Report by:	Simon Michael
Participants:	Alex Suraci, Simon Michael, Scott Lawrence, Daniel Patterson, Daniel Goran
Status:	beta, low activity

http://darcsden.com is a free Darcs (→ 6.5.1) repository hosting service, similar to `patch-tag.com` or (in essence) `github`. The darcsden software is also available (on darcsden) so that anyone can set up a similar service. darcsden is available under BSD license and was created by Alex Suraci.

Alex keeps the service running and fixes bugs, but is mostly focussed on other projects. darcsden has a

clean UI and codebase and is a viable hosting option for smaller projects despite occasional glitches.

The last Hackage release was in 2010. Other committers have been submitting patches, and the darcsden software is close to becoming a just-works installable darcs web ui for general use.

Further reading

http://darcsden.com

6.5.4 darcsum

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

darcsum is an emacs add-on providing an efficient, pcl-cvs-like interface for the Darcs revision control system (→ 6.5.1). It is especially useful for reviewing and recording pending changes.

Simon Michael took over maintainership in 2010, and tried to make it more robust with current Darcs. The tool remains slightly fragile, as it depends on Darcs' exact command-line output, and needs updating when that changes. Dave Love has contributed a large number of cleanups. darcsum is available under the GPL version 2 or later from http://joyful.com/darcsum.

In the last six months darcsum acquired a home page, but there has been little other activity. We are looking for a new maintainer for this useful tool.

Further reading

http://joyful.com/darcsum/

6.5.5 `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 "test", "up", "genpaths", and "doc" subcommands.

Further reading

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

6.6 Deployment

6.6.1 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

We have had two successful Google Summer of Code projects on Cabal this year. Sam Anklesaria worked on a "cabal repl" feature to launch an interactive GHCi session with all the appropriate pre-processing and context from the project's `.cabal` file. Mikhail Glushenkov worked on a feature so that "cabal install" can build independent packages in parallel (not to be confused with building modules within a package in parallel). The code from both projects is available and they are awaiting integration into the main Cabal repository, which we expect to happen over the course of the next few months.

The "cabal test" feature which was developed as a GSoC project last summer has matured significantly in the last 6 months, thanks to continuing effort from Thomas Tuegel and Johan Tibell. The basic test interface will be ready to use in the next release, and there has been some progress on the "detailed" test interface.

The IHG is currently sponsoring some work on cabal-install. The first fruits of this work is a new dependency solver for cabal-install which is now included in the development version. The new solver can find solutions in more cases and produces more detailed error messages when it cannot find a solution. In addition, it is better about avoiding and warning about breaking existing installed packages. We also expect it to be a better basis for other features in future. For more details see the presentation by Andres Löh.

http://haskell.org/haskellwiki/HaskellImplementorsWorkshop/2011/Loeh

The last 6 months has seen significant progress on the new `hackage-server` implementation with help from many new volunteers, in particular Max Bolingbroke,

but also several other people who helped at hackathons and subsequently. The IHG funded Well-Typed to improve package mirroring so that continuous nearly-live mirroring is now possible. We are also grateful to factis research GmbH who have kindly donated a VM to help the hackage developers test the new server code. We expect to do live mirroring and public beta testing using this server during the next few months.

Looking forward

Users are increasingly relying on hackage and cabal-install and are increasingly frustrated by dependency problems. Solutions to the variety of problems do exist. It will however take sustained effort to solve them. The good news is that there is the realistic prospect of the new hackage-server being ready in the not too distant future with features to help monitor and encourage package quality, and the recent work on cabal-install should reduce the frustration level somewhat.

The last 6 months has seen a good upswing in the number of volunteers spending their time on cabal and hackage, so much so that a clear bottleneck is patch review and integration bandwidth. A similar issue is that many of the long standing bugs and feature requests require significant refactoring work which many volunteers feel reluctant or unable to do. Assistance in these areas would be very valuable indeed.

We would like to encourage people considering contributing to join the `cabal-devel` mailing list so that we can increase development discussion and improve collaboration. 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.

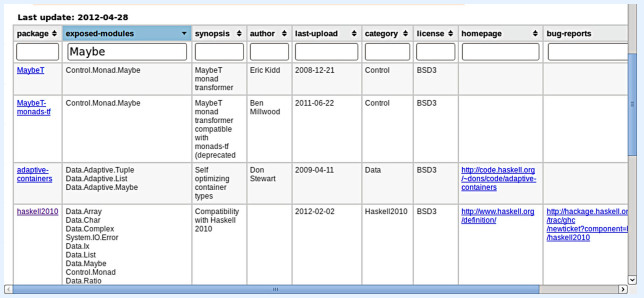
Further reading

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

6.6.2 Portackage — A Hackage Portal

Report by: Andrew G. Seniuk

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



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.

7 Libraries, Applications, Projects

7.1 Language Features

7.1.1 Conduit

Report by: Michael Snoyman
Status: experimental

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 with 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. It is based around the concept of a cursor. In particular, we have sources that can be pulled from and sinks that can be pushed to. There's nothing revolutionary there: this is the same concept powering such low-level approaches as file descriptor I/O. However, we have a few higher-level facilities that make for a simpler usage:

- Monadic composition allows us to combine simpler components into more complicated actions.
- We also have conduits (the namesake of the package), which allow transformations of data. For example, it's trivial to combine a source which reads from a file and a conduit that decompresses data.
- Combined with the resourcet package, we have fully deterministic and exception safe resource handling.

The design space is still not fully resolved. The enumerator approach continues to be used and thrive, and alternatives like pipes are in development as well. The community is currently having a very healthy and lively debate about the merits of each approach. It is likely that we will continue to see improvements and refinements.

Meanwhile, the team behind conduit feels it is ready to be used today. The Web Application Interface (WAI) and Yesod have both moved over to conduit, and have experienced drastic simplification of the code bases. Conduit has also allowed a much simplified HTTP API in the form of http-conduit. In other words, while the package is relatively young, it has already proven vital for our daily workflow, and we believe that many in the community can benefit from it already.

Further reading

- http://www.yesodweb.com/book/conduits
- https://github.com/mezzohaskell/mezzohaskell/blob/master/chapters/libraries/conduit.md

7.1.2 Free Sections

Report by: Andrew G. Seniuk

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

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

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

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

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

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

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

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

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

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

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

Further reading

fremissant.net/freesect

7.2 Education

7.2.1 Holmes, Plagiarism Detection for Haskell

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

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, but is currently unpublished.

The tool will *not* be made available through Hackage, but will be available free of use to lecturers on request. Please contact *J.Hage@uu.nl* for more information.

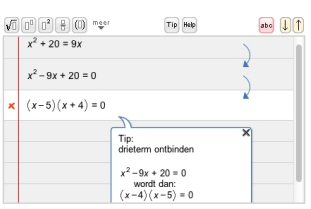
We also have a implemented graph based that computes near graph-isomorphism that seems to work really well in comparing control-flow graphs in an inexact fashion. However, it does not scale well enough in terms of computations to be included in the comparison, and is not mature enough to deal with certain easy refactorings.

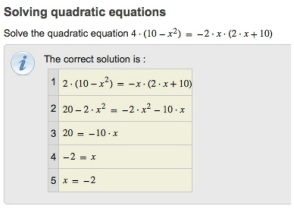
Future work includes a Hare-against-Holmes bash in which Hare users will do their utmost to fool Holmes.

7.2.2 Interactive Domain Reasoners

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

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

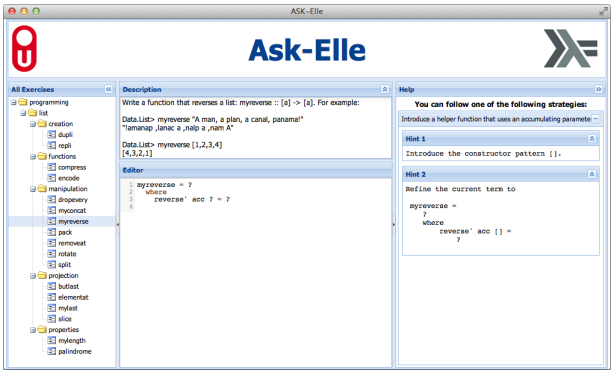




We are adding support for more exercise types, mainly at the level of high school mathematics. For example, our domain reasoner now covers simplifying expressions with exponents, rational equations, and derivatives. We have investigated how users can interleave solving different parts of exercises. We have

extended our strategy language with different combinators for interleaving, and have shown how the interleaving combinators are implemented in the parsing framework we use for recognizing student behavior and providing hints.

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



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

Further reading

- Online exercise assistant (for logic), accessible from our project page.
- Bastiaan Heeren, Johan Jeuring, and Alex Gerdes. Specifying Rewrite Strategies for Interactive Exercises. Mathematics in Computer Science, 3(3):349–370, 2010.
- Bastiaan Heeren and Johan Jeuring. Interleaving Strategies. Conference on Intelligent Computer Mathematics, Mathematical Knowledge Management (MKM 2011).
- Johan Jeuring, Alex Gerdes, and Bastiaan Heeren. A Programming Tutor for Haskell. To appear in Lecture Notes Central European School on Functional Programming, (CEFP 2011). Try our tutor at http://ideas.cs.uu.nl/ProgTutor/.

7.3 Parsing and Transforming

7.3.1 The grammar-combinators Parser Library

Report by:	Dominique Devriese
Status:	partly functional

The grammar-combinators library is an experimental parser library written in Haskell (LGPL license). The library features much of the power of a parser generator like Happy or ANTLR, but with the library approach and most of the benefits of a parser combinator library.

The project's initial release was in September 2010. A paper about the main idea has been presented at the PADL'11 conference and an accompanying technical report with more implementation details is available online. The library is published on Hackage under the name grammar-combinators.

The library works with an explicit, typed representation of non-terminals, allowing fundamentally more powerful grammar algorithms, including various grammar analysis, transformation and pretty-printing libraries etc. A disadvantage is that higher-order combinators modelling recursive concepts like many and some require more work to write. The library is currently not yet suited for mainstream use. Performance is not ideal and many real-world features are missing. People interested to work on these topics are very welcome to contact us!

Further reading

http://projects.haskell.org/grammar-combinators/

7.3.2 epub-metadata

Report by:	Dino Morelli
Status:	stable, actively developed

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

7.3.3 Utrecht Parser Combinator Library: uu-parsinglib

Report by:	Doaitse Swierstra
Status:	actively developed

The previous extension for recognizing merging parsers was generalized so now any kind of applicative and monadic parsers can be merged in an interleaved way. As an example take the situation where many different programs write log entries into a log file, and where each log entry is uniquely identified by a transaction

number (or process number) which can be used to distinguish them. E.g., assume that each transaction consists of an a, a b and a c action, and that a digit is used to identify the individual actions belonging to the same transaction; the individual transactions can now be recognized by the parser:

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

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

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

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

Features

- Much simpler internals than the old library (http://haskell.org/communities/05-2009/html/report.html#sect5.5.8).
- Combinators for easily describing parsers which produce their results online, do not hang on to the input and provide excellent error messages. As such they are "surprise free" when used by people not fully aware of their internal workings.
- Parsers "correct" the input such that parsing can proceed when an erroneous input is encountered.
- The library basically provides the to be preferred applicative interface and a monadic interface where this is really needed (which is hardly ever).
- No need for *try*-like constructs which makes writing `Parsec` based parsers tricky.
- Scanners can be switched dynamically, so several different languages can occur intertwined in a single input file.
- Parsers can be run in an interleaved way, thus generalizing the merging and permuting parsers into a single applicative interface. This makes it e.g. possible to deal with white space or comments in the input in a completely separate way, without having to think about this in the parser for the language at hand (provided of course that white space is not syntactically relevant).

Future plans

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

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

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

Contact

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

7.3.4 Regular Expression Matching with Partial Derivatives

Report by:	Martin Sulzmann
Participants:	Kenny Zhuo Ming Lu
Status:	stable

We are still improving the performance of our matching algorithms. The latest implementation can be downloaded via hackage.

Further reading

- http://hackage.haskell.org/package/regex-pderiv
- http://sulzmann.blogspot.com/2010/04/regular-expression-matching-using.html

7.3.5 regex-applicative

Report by:	Roman Cheplyaka
Status:	active development

regex-applicative is aimed to be an efficient and easy to use parsing combinator library for Haskell based on regular expressions.

There are several ways in which one can specify what part of the string should be matched: the whole string, a prefix or an arbitrary part ("leftmost infix") of the string.

Additionally, for prefix and infix modes, one can demand either the longest part, the shortest part or the first (in the left-biased ordering) part.

Finally, other things being equal, submatches are chosen using left bias.

Recently the performance has been improved by using more efficient algorithm for the parts of the regular expression whose result is not used.

Example code can be found on the wiki.

Further reading

- http://hackage.haskell.org/package/regex-applicative
- http://github.com/feuerbach/regex-applicative

7.4 Generic and Type-Level Programming

7.4.1 Unbound

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

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

Further reading

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

7.4.2 FlexiWrap

Report by:	Iain Alexander
Status:	experimental

A library of flexible newtype wrappers which simplify the process of selecting appropriate typeclass instances, which is particularly useful for composed types.

Version 0.1.0 has been released on Hackage, providing support for a more comprehensive range of typeclasses when wrapping simple values, and some documentation. Work is still ongoing to flesh out the typeclass instances available and improve the documentation.

7.4.3 Generic Programming at Utrecht University

Report by:	José Pedro Magalhães
Participants:	Johan Jeuring, Sean Leather
Status:	actively developed

See: http://www.haskell.org/communities/11-2010/html/report.html#sect8.5.3.

7.4.4 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.

Further reading

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

7.5 Proof Assistants and Reasoning

7.5.1 HERMIT

Report by:	Andy Gill
Participants:	Andy Gill, Andrew Farmer, Ed Komp, Neil Sculthorpe
Status:	active

The Haskell Equational Reasoning Model-to-Implementation Tunnel (HERMIT) is an NSF-funded project being run at KU (→ 9.11), which aims to improve the applicability of Haskell-hosted Semi-Formal Models to High Assurance Development. Specifically, HERMIT will use: a Haskell-hosted DSL; the Worker/Wrapper Transformation; and a new refinement user interface to perform rewrites directly on Haskell Core, the GHC internal representation.

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

The project started in Spring 2012, and is expected to run for two years. Neil Sculthorpe, who got his PhD from the University of Nottingham in 2011, has joined as a senior member of the project, and Andrew Farmer and Ed Komp round out the team. We have already reworked the KURE DSL (http://www.haskell.org/communities/11-2008/html/report.html#sect5.5.7) as the basis of our rewrite capabilities, and constructed the rewrite kernel. The entire system uses the GHC plugin architecture, and we have small examples successfully being transformed through a simple REPL. A web-based API is being constructed, and an Android version is planned. We aim to write up a detailed introduction of our architecture and implementation for the Haskell Symposium.

Further reading

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

7.5.2 Automated Termination Analyzer for Haskell

Report by:	Jürgen Giesl
Participants:	Matthias Raffelsieper, Peter Schneider-Kamp, Stephan Swiderski, René Thiemann
Status:	actively developed

See: http://www.haskell.org/communities/05-2011/html/report.html#sect7.6.1.

7.5.3 HTab

Report by:	Guillaume Hoffmann
Status:	active development

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

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

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

Further reading

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

7.5.4 Free Theorems for Haskell

Report by:	Janis Voigtländer
Participants:	Daniel Seidel

Free theorems are statements about program behavior derived from (polymorphic) types. Their origin is the polymorphic lambda-calculus, but they have also been applied to programs in more realistic languages like Haskell. Since there is a semantic gap between the original calculus and modern functional languages, the underlying theory (of relational parametricity) needs to be refined and extended. We aim to provide such new theoretical foundations, as well as to apply the theoretical results to practical problems. The research grant that sponsored Daniel's position has been extended for another round of funding. However, currently we are both consumed by teaching the (by local definition, imperative) programming intro course here at U Bonn, in C (yes, in C), plus an advanced functional programming course, in Haskell.

On the practical side, we maintain a library and tools for generating free theorems from Haskell types, originally implemented by Sascha Böhme and with contributions from Joachim Breitner and now Matthias Bartsch. Both the library and a shell-based tool are available from Hackage (as free-theorems and ftshell,

respectively). There is also a web-based tool at http://www-ps.iai.uni-bonn.de/ft/. Features include:

- three different language subsets to choose from
- equational as well as inequational free theorems
- relational free theorems as well as specializations down to function level
- support for algebraic data types, type synonyms and renamings, type classes
- plain text, LaTeX source, PDF, and inline graphics output with nicely typeset theorems

Further reading

http://www.iai.uni-bonn.de/~jv/ft-project/

7.5.5 Streaming Component Combinators

Report by:	Mario Blažević
Status:	experimental, actively developed

See: http://www.haskell.org/communities/11-2010/html/report.html#sect9.6.5.

7.5.6 Swish

Report by:	Douglas Burke
Participants:	Graham Klyne, Vasili I Galchin
Status:	experimental

Swish is a framework for performing deductions in RDF data using a variety of techniques. Swish is conceived as a toolkit for experimenting with RDF inference, and for implementing stand-alone RDF file processors (usable in similar style to CWM, but with a view to being extensible in declarative style through added Haskell function and data value declarations). It explores Haskell as "a scripting language for the Semantic Web", is a work-in-progress, and currently incorporates:

- Support for Turtle, Notation3, and NTriples formats.
- RDF graph isomorphism testing and merging.
- Display of differences between RDF graphs.
- Inference operations in forward chaining, backward chaining and proof-checking modes.
- Simple Horn-style rule implementations, extendable through variable binding modifiers and filters.
- Class restriction rule implementation, primarily for datatype inferences.
- RDF formal semantics entailment rule implementation.
- Complete, ready-to-run, command-line and script-driven programs.

Current Work

A number of incremental changes have been made to the code base, including support for version 7.2 of GHC and some minor optimisations. A parser and formatter for the Turtle format were added, the API changed to use the Text datatype where appropriate, and the vocabulary module was extended to include terms from the Dublin Core, FOAF, Geo and SIOC vocabularies.

Future plans

Continue the clean up and replacement of code with packages from Hacakge. Look for commonalities with the other existing RDF Haskell package, `rdf4h`. Community input — whether it be patches, new code or just feature requests — are more than welcome.

Further reading

- https://bitbucket.org/doug_burke/swish/
- http://www.ninebynine.org/RDFNotes/Swish/Intro.html
- http://protempore.net/rdf4h/

7.6 Mathematical Objects

7.6.1 normaldistribution: Minimum Fuss Normally Distributed Random Values

Report by:	Björn Buckwalter
Status:	stable

Normaldistribution is a new package that lets you produce normally distributed random values with a minimum of fuss. The API builds upon, and is largely analogous to, that of the Haskell 98 *Random* module (more recently *System.Random*). Usage can be as simple as:

$$sample \leftarrow normalIO$$

For more information and examples see the package description on Hackage.

Further reading

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

7.6.2 dimensional: Statically Checked Physical Dimensions

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

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

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

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

Ongoing experimental work includes:

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

$$v :: \mathit{Fractional}\ a \Rightarrow \mathit{Time}\ a \to \mathit{Velocity}\ a$$
$$v\ t = \mathit{diff}\ d\ t$$

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

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

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

Further reading

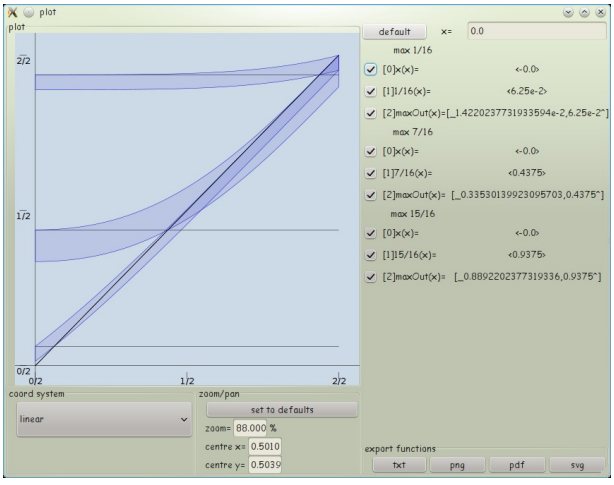
- http://dimensional.googlecode.com
- https://github.com/bjornbm/dimensional-vectors
- https://github.com/bjornbm/dimensional-experimental
- http://flygdynamikern.blogspot.com/2012/02/announce-dimensional-tf-010-statically.html

7.6.3 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 and fast arbitrary precision correctly rounded **interval arithmetic**, including both standard and inverted intervals with Kaucher arithmetic
- arbitrary precision arithmetic of **polynomial intervals** (similar to but more general than Taylor Models) to
  - automatically reduce overestimations in interval computations
  - efficiently support validated numerical integration, specifically in the simulation of ordinary differential equation (ODE) and hybrid system initial value problems (IVPs)
  - automatically decide 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
  - ability to increase computational effort to reduce the effect of rounding and partiality, converging to exact operations and total 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
- 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 Summer 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 a Haskell-only implementation in Summer 2012. The development files include demos that solve selected ODE and hybrid system IVPs using polynomial intervals.

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

Further reading

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

7.6.4 Paraiso

Report by:	Takayuki Muranushi
Status:	active development

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

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

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

Further reading

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

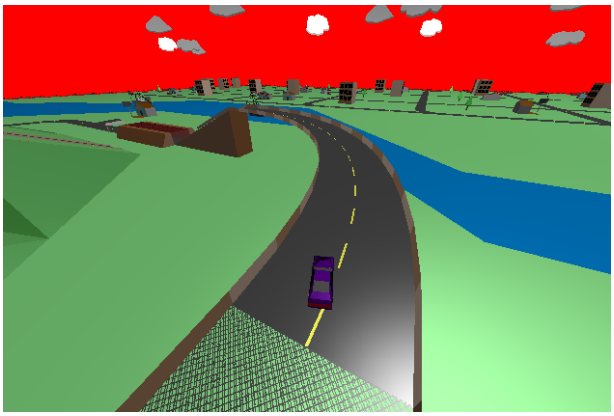
7.6.5 Bullet

Report by:	Csaba Hruska
Status:	experimental, active development

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

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

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



Further reading

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

7.7 Data Types and Data Structures

7.7.1 HList — A Library for Typed Heterogeneous Collections

Report by:	Oleg Kiselyov
Participants:	Ralf Lämmel, Keean Schupke, Gwern Branwen

HList is a comprehensive, general purpose Haskell library for typed heterogeneous collections including extensible polymorphic records and variants. HList is analogous to the standard list library, providing a host of various construction, look-up, filtering, and iteration

primitives. In contrast to the regular lists, elements of heterogeneous lists do not have to have the same type. HList lets the user formulate statically checkable constraints: for example, no two elements of a collection may have the same type (so the elements can be unambiguously indexed by their type).

An immediate application of HLists is the implementation of open, extensible records with first-class, reusable, and compile-time only labels. The dual application is extensible polymorphic variants (open unions). HList contains several implementations of open records, including records as sequences of field values, where the type of each field is annotated with its phantom label. We and others have also used HList for type-safe database access in Haskell. HList-based Records form the basis of OOHaskell. The HList library relies on common extensions of Haskell 2010. HList is being used in AspectAG (→ 5.3.2), typed EDSL of attribute grammars, and in HaskellDB.

The October 2011 version of HList library has many changes, mainly related to deprecating `TypeCast` (in favor of `~`) and getting rid of overlapping instances. The only use of OverlappingInstances is in the implementation of the generic type equality predicate `TypeEq`. We plan to remove even that remaining single occurrence. The code works with GHC 7.0.4.

Future plans include the implementation of TypeEq without resorting to overlapping instances (so, HList will be overlapping-free), and moving towards type functions and expressive kinds.

Further reading

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

7.7.2 Persistent

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

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

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

$$selectList\ [PersonFirstName == .\ \texttt{"Simon"},\\ PersonLastName == .\ \texttt{"Jones"}]\ []$$

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

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

Future plans

Future directions for Persistent:

- Full CouchDB support
- A MongoDB specific query layer
- Adding key-value databases like Redis without a query layer.

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

Further reading

http://yesodweb.com/book/persistent

7.7.3 DSH — Database Supported Haskell

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

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).

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.

Support for arbitrary data types. In Haskell, the creation of new data types using `data` provides *the* means to model user-defined objects. We are currently working on support for arbitrary data types in DSH such that these user-defined types may be queried just like the supported built-in Haskell types. This work rests on GHC's new generic deriving mechanism.

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.

Further reading

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

7.8 User Interfaces

7.8.1 Gtk2Hs

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

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

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

Gtk2Hs features:

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

The most recent 0.12.3 release widens the variety of ecosystems that can build Gtk2Hs by supporting GHC 7.4 and improving the Windows support, includes bindings to a few overlooked Gtk behaviors for restoring widget properties to their defaults, and sports various additional bugfixes and documentation improvements.

Further reading

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

7.8.2 xmonad

Report by:	Gwern Branwen
Status:	active development

XMonad is a tiling window manager for X. Windows are arranged automatically to tile the screen without gaps or overlap, maximizing screen use. Window manager features are accessible from the keyboard; a mouse

is optional. XMonad is written, configured, and extensible in Haskell. Custom layout algorithms, key bindings, and other extensions may be written by the user in config files. Layouts are applied dynamically, and different layouts may be used on each workspace. Xinerama is fully supported, allowing windows to be tiled on several physical screens.

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

Details of changes between releases can be found in the release notes:

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

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

Further reading

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

7.9 Functional Reactive Programming

7.9.1 reactive-banana

Report by:	Heinrich Apfelmus
Status:	active development

Reactive-banana is a practical library for functional reactive programming (FRP).

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

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

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

- Programmers interested in implementing FRP will have a *reference* for a *simple semantics* with a working implementation. The library stays close to the semantics pioneered by Conal Elliott.

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

Status. Version 0.6.0.0 of the reactive-banana library will shortly be released on Hackage. It provides a solid push-based implementation.

Compared to the previous report, the API has been refined, making the library ever more pleasant to use. The internals have been rewritten completely to prepare for the introduction of dynamic event switching in a future version.

Also, I have been approached by Mathijs Kwik who desired to use functional reactive programming in conjunction with the JavaScript backend of the Utrecht

Haskell Compiler (UHC). Consequently, I have modified the library and the latest version can now be compiled with UHC. In other words, it has now become possible to use FRP with Haskell in the web browser.

Current development focuses on the implementation of dynamic event switching. Examples from computer music are planned.

Notable examples. 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
- Cabal package: http://hackage.haskell.org/package/reactive-banana
- Developer blog: http://apfelmus.nfshost.com/blog.html
- reactive-balsa: http://www.haskell.org/haskellwiki/Reactive-balsa

7.9.2 Functional Hybrid Modelling

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

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

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

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

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

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

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

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

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

Further reading

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

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

7.9.3 Elerea

Report by:	Patai Gergely
Status:	experimental, active

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

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

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

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

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

Further reading

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

7.10 Graphics

7.10.1 LambdaCube

Report by:	Csaba Hruska
Status:	experimental, active development

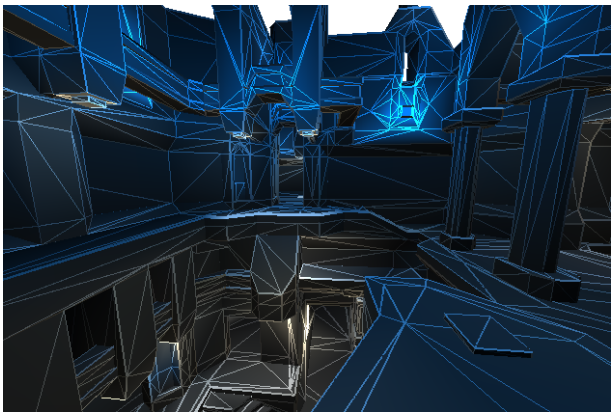
LambdaCube is a 3D graphics library entirely written in Haskell.

The main goal of this project is to provide a modern and feature rich graphical backend for various Haskell projects, and in the long run it is intended to be a practical solution even for serious purposes. With LambdaCube we can program the GPU in a purely functional manner, just like with GPipe, but LambdaCube provides much better runtime performance.

Over the last few months, the library has been completely rewritten. 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.

Our mid term plan is to define a standalone DSL, so the graphics pipeline could be dynamically reprogrammed during runtime. Using our graphics language, we can implement arbitrary rendering techniques in a hardware independent and compositional way. All resource handling and performance optimizations will be done by the graphics backend. Currently we are targeting OpenGL 3.3, but OpenGL ES support is also planned.

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

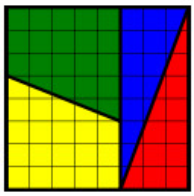
- https://github.com/csabahruska/lc-dsl

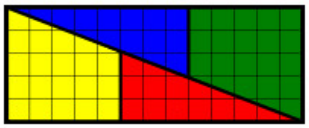
- http://www.haskell.org/haskellwiki/LambdaCubeEngine
- http://hackage.haskell.org/package/stunts
- http://www.youtube.com/watch?v=kDu5aCGc8l4

7.10.2 diagrams

Report by:	Brent Yorgey
Participants:	Peter Hall, Andy Gill, Deepak Jois, Ian Ross, 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.

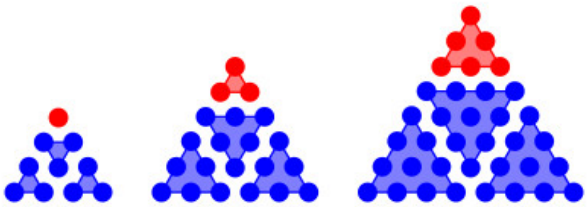




Since the previous HCAR, a new version of the framework has been released, featuring experimental support for animations; a new package of user-contributed modules, so far including tree drawing, Apollonian gaskets, planar tilings, "wrapped" layout, and turtle graphics; better performance; many other small additions and improvements; and a redesigned website.

There is also a growing diagrams "ecosystem"; related projects under development include TikZ and HTML5 canvas backends, a Logo interpreter, a graphing application, and a framework for creating interactive GTK/cairo applications.

There is plenty more work to be done; new contributors are welcome!



Future plans

A native SVG backend is under active development and targeted for the next release of the framework. The cairo backend will still be supported, but SVG will replace cairo as the default "out-of-the-box" backend, vastly simplifying installation for new useres. Other plans for the near future include support for drawing arrows and improvements to the handling of named subdiagrams. Longer-term plans include support for interactive diagrams, a custom Gtk application for editing diagrams, and any other awesome stuff we think of.

Further reading

- http://projects.haskell.org/diagrams
- http://projects.haskell.org/diagrams/gallery.html
- http://code.google.com/p/diagrams/issues/list

7.11 Audio

7.11.1 Audio Signal Processing

Report by:	Henning Thielemann
Status:	experimental, active development

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

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

Recent advances are:

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

Further reading

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

7.11.2 Live-Sequencer

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

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

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

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

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

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

Future plans

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

Further reading

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

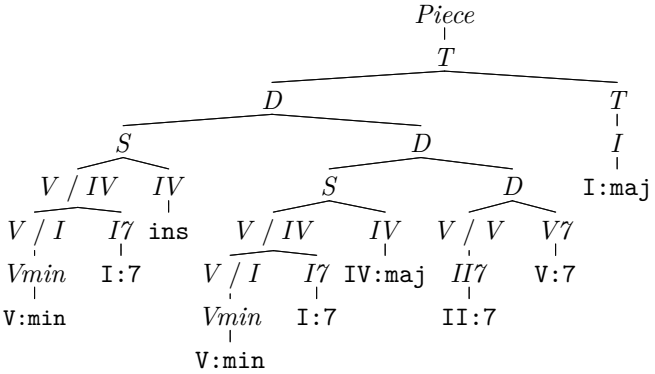
7.11.3 Functional Modelling of Musical Harmony

Report by:	José Pedro Magalhães
Participants:	W. Bas de Haas
Status:	actively developed

Music theory has been essential in composing and performing music for centuries. Within Western tonal music, from the early Baroque on to modern-day jazz and pop music, the function of chords within a chord sequence can be explained by harmony theory. Although Western tonal harmony theory is a thoroughly studied area, formalising this theory is a hard problem.

We have developed a system, named HarmTrace, that formalises the rules of tonal harmony as a Haskell (generalized) algebraic datatype. Given a sequence of chord labels, the harmonic function of a chord in its tonal context is automatically derived. For this, we use several advanced functional programming techniques, such as type-level computations, datatype-generic programming, and error-correcting parsers. We have an experience report at ICFP'11 detailing this project.

As an example, we show a tree representation of the harmony analysis of a short music fragment:



This tree is a visual representation of a value of a Haskell datatype encoding musical harmony, with common notions such as tonic, dominant, etc. Such trees are generated from input sequences of chord labels such as `C:maj F:maj G:7 C:Maj`.

A functional model of harmony offers various benefits: for instance, it can help musicologists in batch-analysing large corpora of digitised scores, but it has proven to be especially useful for solving Music Information Retrieval (MIR) problems. MIR is the research field that aims to provide methods that keep large collections of digital music accessible and maintainable. Hence, besides generating musically meaningful harmonic analyses, HarmTrace explores ways of exploiting these generated analyses to improve the similarity assessment of chord sequences and the automatic extraction for chord labels from musical audio. Some empirical evidence showing that the harmonic analyses

of HarmTrace improve harmonic similarity estimation has been published at the International Society for Music Information Retrieval conference 2011.

The code is also available on Hackage.

Further reading

http://www.cs.uu.nl/wiki/GenericProgramming/HarmTrace

7.12 Text and Markup Languages

7.12.1 HaTeX

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

Description

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

From a global sight, it's composed of:

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

What is new?

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

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

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

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

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

Future plans

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

Contact

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

`hatex <at> projects.haskell.org`

Of course, you always can mail to the maintainer.

Further reading

- HaTeX project page: http://dhelta.net/hprojects/HaTeX

7.12.2 Haskell XML Toolbox

Report by:	Uwe Schmidt
Status:	seventh major release (current release: 9.2)

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 still to be done, so the first release will be in May or June this year. The implementation will be rather complete, except the datatype library for XML Schema. Some of the time and date types are not yet included. With the next HXT release the master thesis will be published on the HXT homepage.

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.

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

7.12.3 epub-tools (Command-line epub Utilities)

Report by:	Dino Morelli
Status:	stable, actively developed

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

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

Further reading

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

7.13 Hardware Design

7.13.1 CλaSH

Report by:	Christiaan Baaij
Participants:	Arjan Boeijink, Jan Kuper, Anja Niedermeier, Matthijs Kooijman, Marco Gerards
Status:	experimental

See: http://www.haskell.org/communities/05-2011/html/report.html#sect7.5.1.

7.13.2 Kansas Lava

Report by:	Andy Gill
Participants:	Andy Gill, Andrew Farmer, Ed Komp, Bowe Neuenschwander, Garrin Kimmell (University of Iowa)
Status:	ongoing

Kansas Lava is a Domain Specific Language (DSL) for expressing hardware descriptions of computations, and is hosted inside the language Haskell. Kansas Lava programs are descriptions of specific hardware entities, the connections between them, and other computational abstractions that can compile down to these entities. Large circuits have been successfully expressed using Kansas Lava, and Haskell's powerful abstraction mechanisms, as well as generic generative techniques, can be applied to good effect to provide descriptions of highly efficient circuits. Kansas Lava draws considerably from Xilinx Lava (http://www.haskell.org/communities/11-2010/html/report.html#sect3.7) and Chalmers Lava (→ 9.10).

The release of Kansas Lava, version 0.2.4, happened in early November. Based round this release, there are a number of resources for users, including a (draft)

tutorial, and a youtube channel with walkthroughs of our Lava in use.

On top of Kansas Lava, we are developing Kansas Lava Cores, which was released on hackage at the same time as Kansas Lava. 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 an emulator for the Spartan3e.

Using various components provided as Kansas Lava Cores, we are developing the λ-bridge (http://www.haskell.org/communities/11-2011/html/report.html#sect8.8.2), with implementations in Haskell and Kansas Lava of a simple protocol stack for communicating with FPGAs. We have early prototypes working, and implementation in Kansas Lava continues.

Finally, we are working on a Lava idiom called a `Patch`, which is a Kansas Lava component interface that uses types to declare protocols and handshakes needed and used. Most of components in the Kansas Lava Cores are instances of our `Patch` idiom. There is a PADL 2012 paper describing `Patch`, including the design and implementation of a controller for an ST7066U-powered LCD display.

Further reading

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

7.14 Natural Language Processing

7.14.1 NLP

Report by: Eric Kow

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

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

Recently released packages and projects

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

New packages and projects in development

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

At the present, the mailing list is mainly used to make announcements to the Haskell NLP community. At the time of this writing, there is an ongoing Coursera online NLP class, for which some of list members have expressed an interest in doing the assingments in Haskell. We hope that we will continue to expand the list and expand our ways of making it useful to people potentially using Haskell in the NLP world.

Further reading

http://projects.haskell.org/nlp

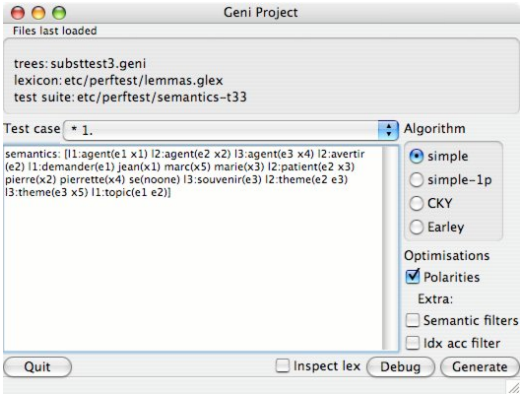
7.14.2 GenI

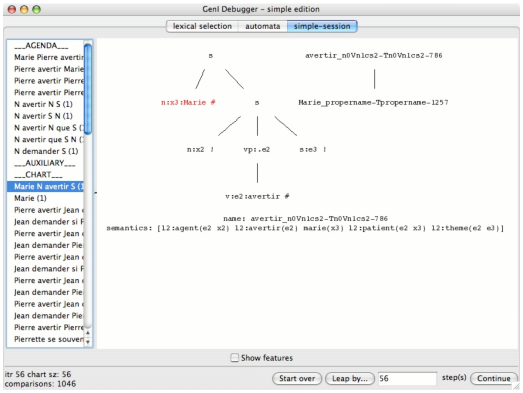
Report by: Eric Kow

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.

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





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

Further reading

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

7.15 Others

7.15.1 leapseconds-announced

Report by:	Björn Buckwalter
Status:	stable, maintained

The leapseconds-announced library provides an easy to use static LeapSecondTable with the leap seconds announced at library release time. It is intended as a quick-and-dirty leap second solution for one-off analyses concerned only with the past and present (i.e. up until the next as of yet unannounced leap second), or for applications which can afford to be recompiled against an updated library as often as every six months.

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

Further reading

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

7.15.2 FunGEn

Report by:	Simon Michael
Status:	usable; ready for contributors and users

FunGEn (Functional Game Engine) is a BSD-licensed cross-platform 2D game engine implemented in and for Haskell, using OpenGL and GLUT. It was created in 2002 by Andre Furtado, updated in 2008 by Simon Michael and Miloslav Raus, and revived again in 2011, with a GHC 6.12-tested 0.3 release on Hackage, preliminary haddockification and a new home repo.

FunGEn remains the quickest path to building cross-platform graphical games in Haskell, due to its convenient game framework and widely-available dependencies. It comes with several working examples that are quite easy to read and build (pong, worms). In the last six months there has been little activity and a new maintainer would be welcome.

FunGEn-related discussions most often appear in the #haskell-game channel on irc.freenode.net.

Further reading

http://darcsden.com/simon/fungen

7.15.3 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 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. More recently, we have started to work on extending the language to deal with more system-level aspects such as memory layout and concurrency.

The implementation is available from Hackage.

Further reading

- http://feldspar.inf.elte.hu
- http://hackage.haskell.org/package/feldspar-language
- http://hackage.haskell.org/package/feldspar-compiler

7.15.4 ADPfusion

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

ADPfusion combines ideas of Algebraic Dynamic Programming (ADP), established by Robert Giegerich et al., and stream-fusion by Coutts et al. into a high-level framework to optimize the run-time performance of algorithms based on context-free grammars with a special emphasis on applications in computational biology.

As with ADP, we aim for a separation of concerns. While an algorithm implementing a, say, context-free grammar (CFG) typically leads to an intertwining of several "concerns" (search space, evaluation, tabulation, optimization of code), we want to separate them as much as possible to achieve clarity of code while retaining an efficient implementation.

ADP separates the search space which contains all possible solutions from the evaluation of each candidate. This idea allows for a very clear description of an algorithm, to replace evaluation strategies, or even combine them.

Deforestation and stream-fusion are well-established ideas to improve the performance of (low-level) list- and vector-based algorithms in Haskell.

ADPfusion combines both ideas, allowing for a high-level description of any CFG, with performance coming close to that of optimized `C`. The library can easily be adapted to handle special data structures, and will be extended to allow for more general grammars than just CFG's, as well as table designs with more than two dimensions.

A tutorial is in preparation, the library and two examples are available on hackage.

Further reading

- http://www.tbi.univie.ac.at/~choener/adpfusion
- http://hackage.haskell.org/package/ADPfusion
- http://hackage.haskell.org/package/Nussinov78
- http://hackage.haskell.org/package/RNAFold

7.15.5 Biohaskell

Report by:	Ketil Malde
Participants:	Christian Höner zu Siederdissen, Nick Ignolia, Felipe Almeida Lessa

Bioinformatics in Haskell is a steadily growing field, and the *Bio* section on Hackage now contains 45 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.

Bioinformatics is a diverse field, and consequently, we have different libraries covering mostly separate areas.

Recently, the *biolib library* is being split up into smaller, standalone packages, which along with other contributions, depend on the small biocore library for some standard data types and definitions.

Among new contributions are Christian's ADPfusion stuff (→ 7.15.4), combining Robert Giegerich's Algebraic Dynamic Programming with stream-fusing combinators.

Further reading

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

7.15.6 hledger

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

hledger is a library and end-user tool (with command-line, curses and web interfaces) for converting, recording, and analyzing financial transactions, using a simple human-editable plain text file format. It is a haskell port and friendly fork of John Wiegley's Ledger, licensed under GNU GPLv3+.

hledger aims to be a reliable, practical tool for daily use. It reports charts of accounts or account balances, filters transactions by type, helps you record new transactions, converts CSV data from your bank, publishes your text journal with a rich web interface, generates simple charts, and provides an API for use in your own financial scripts and apps.

In the last six months there have been two major releases. 0.15 focussed on features and 0.16 focussed on quality. Changes include:

- new modal command-line interface, extensible with hledger-* executables in the path
- more useful web interface, with real account registers and basic charts
- hledger-web no longer needs to create support files, and uses latest yesod & warp
- more ledger compatibility
- misc command enhancements, API improvements, bug fixes, documentation updates
- lines of code increased by 3k to 8k
- project committers increased by 6 to 21

Current plans include:

- Continue the release rhythm of odd-numbered = features, even-numbered = quality/stability/polish, and releasing on the first of a month
- In 0.17, clean up the storage layer, allow rcs integration via filestore, and read (or convert) more formats
- Keep working towards wider usefulness, improving the web interface and providing standard financial reports

Further reading

http://hledger.org

7.15.7 sshtun (Wrapper daemon to manage an ssh tunnel)

Report by:	Dino Morelli
Status:	experimental, actively developed

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

sshtun is available from Hackage and the Darcs repository below.

Further reading

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

7.15.8 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 previous stable release targetted the XMLRPC Mollom API, the new releases will all target the REST API.

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

The next step is to wrap hMollom for use in the Haskell web frameworks, for example Snap and Yesod, so people can add it to their websites and have comment spam filtered out.

Further reading

http://github.com/itkovian/hMollom

7.15.9 Galois Open-Source Projects on GitHub

Report by:	Jason Dagit
Status:	active

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

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

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

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

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

8 Commercial Users

8.1 Well-Typed LLP

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

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

We are continuing to grow, with an additional contractor since the last HCAR. While we aren't currently hiring, we have a number of interesting possibilities on the horizon, so we're always happy to receive CVs.

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). Most visibly, since the last HCAR, we have put out the 7.2.2 and 7.4.1 releases, and are currently working on the 7.4.2 release. We expect that the next major release, 7.6.1, will be around the time of the next HCAR, in about 6 months time.

We also continue to coordinate and do work for the Industrial Haskell Group (IHG) (→ 8.3). Recently, the work has focussed on making a 64bit Windows port of GHC. We expect this to be released as part of GHC 7.6.1.

Within the Parallel GHC Project (→ 5.1.3), we have been helping our partners to implement parallel, concurrent and distributed software in Haskell, and working to improve the tools and libraries, such as ThreadScope. We've also started development of Cloud Haskell, an Erlang-like system for Haskell.

In addition, we remain quite involved in the community, maintaining several packages on Hackage. We have also continued to evangelise Haskell, including presenting talks at FP eXchange and a number of universities. Many of us were also at the Utrecht hackathon, where we worked on a number of projects, including Cabal, the pipes libraries, Cloud Haskell and some GHC extensions.

We expect to continue to be involved over the coming months, and in particular several of us plan to be in Copenhagen in September for CUFP, the Haskell Symposium and ICFP. Please get in touch if you'd like to meet up with us there.

We are of course always looking for new clients and projects, too, so if you are interested in hiring us, just drop us an e-mail.

Further reading

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

8.2 Bluespec Tools for Design of Complex Chips and Hardware Accelerators

Report by: Rishiyur Nikhil
Status: commercial product

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

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

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

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

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

Status and availability

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

Further reading

- *Abstraction in Hardware System Design*, R.S. Nikhil, in *Communications of the ACM*, 54:10, October 2011, pp. 36-44.
- *Bluespec, a General-Purpose Approach to High-Level Synthesis Based on Parallel Atomic Transactions*, R.S. Nikhil, in *High Level Synthesis: from Algorithm to Digital Circuit, Philippe Coussy and Adam Morawiec (editors)*, Springer, 2008, pp. 129-146.
- *BSV by Example*, R.S. Nikhil and K. Czeck, 2010, book available on Amazon.com.
- http://bluespec.com/SmallExamples/index.html: from *BSV by Example*.
- http://www.cl.cam.ac.uk/~swm11/examples/bluespec/: Simon Moore's BSV examples (U. Cambridge).
- http://csg.csail.mit.edu/6.375: *Complex Digital Systems*, MIT courseware.
- http://www.bluespec.com/products/BluDACu.htm: A fun example with many functional programming features — BluDACu, a parameterized Bluespec hardware implementation of Sudoku.

8.3 Industrial Haskell Group

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

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

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

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

In the past six months, the collaborative development scheme funded work on cabal-install, improvements to the Hackage server as well as work on a Win64 port of GHC.

The work on cabal-install has culminated in a new, modular dependency solver that is now released as a part of cabal-install-0.14.0. If you are using the new cabal-install together with a recent GHC, you will get the new solver component by default.

The work on the new Hackage server is still ongoing. A preview of the new system is now continuously running at http://hackage.factisresearch.com/ — this is an automatic mirror of the original Hackage at the moment; you cannot and should not upload new packages to this instance yet.

The Win64 port is currently the main focus of IHG work. At the time of writing this report, registerised (optimized) builds are already working, and we are trying to get ghci up and running, too.

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.

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

Further reading

- http://industry.haskell.org/
- http://industry.haskell.org/status/
- http://hackage.haskell.org/package/cabal-install-0.14.0/
- http://hackage.factisresearch.com/

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 six and a half 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. It is the fact that the DSL is itself functional which has made developing all these tools much easier.

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 New York, 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. 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 (→ 7.3.3).

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. Currently, Oblomov Systems is working 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 madvertise Mobile Advertising

Report by: Adam Drake

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

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

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

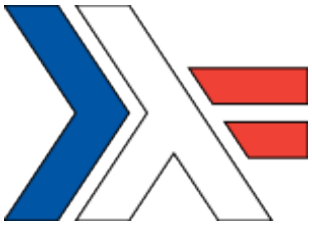
Further reading

http://madvertise.com

9 Research and User Groups

9.1 A French community for Haskell

Report by:	Alp Mestanogullari
Participants:	Valentin Robert, Fabien Georget, and others
Status:	ongoing



During the past few months, we have seen many new Haskellers in the French-speaking communities. Aside from this, Valentin Robert has published a translation of *Learn You a Haskell For Great Good* in French (see the further reading section). It seems Haskell is finally getting more interest from French developers and that is the reason why we are now trying to create some activity around this.

We are currently working on the basics:

- getting an adequate wiki/website,
- figuring out some project ideas, may they be documentation or software projects,
- working on a first French Hackathon event for French Haskellers to meet and get to know each other.

Among us, we happen to have people interested in many areas (Haskell for web programming, high performance Haskell, etc.) so on the long-term we may be able to provide resources about various topics. Another possible idea would be to have some kind of workgroups that would work on a given project, or even do bug hunting for an already existing project. And we have many other ideas, but it will depend on how the community's activity grows. Our priorities are the website and the first Hackathon, that may happen in June in Strasbourg. This is yet to be confirmed.

We warmly welcome anyone interested in helping us create this community! There are all kinds of tasks to accomplish so you do not need to be a Haskell guru to contribute. We also welcome any French-speaking haskellers or even functional programmers to join us either on the IRC channel `#haskell-fr` on Freenode or on the mailing list.

Further reading

- Homepage: http://www.haskell.fr/
- LYAH in French: http://lyah.haskell.fr/
- Haskell-fr mailing list: http://www.haskell.org/mailman/listinfo/haskell-fr
- Original announcement: http://tinyurl.com/3o9tatf

9.2 Haskell at Eötvös Loránd University (ELTE), Budapest

Report by:	PÁLI Gábor János
Status:	ongoing

Education

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

- Programming for first-year BSc students using Haskell, it is officially in the curriculum. It is also taught for foreign language students as part of their program.
- Advanced functional programming using Haskell, it is an optional course for BSc and MSc students.
- Programming in Agda as an optional course for BSc and MSc students.
- Other Haskell-related courses on Lambda Calculus, Type Theory and Implementation of Functional Languages.

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

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

Further reading

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

9.3 Functional Programming at UFMG and UFOP

Report by:	Carlos Camarão
Participants:	Marco Gontijo, Lucília Figueiredo, Rodrigo Ribeiro, Cristiano Vasconcellos, Elton Ribeiro
Status:	active development

The Functional Programming groups at Universidade Federal de Minas Gerais and Universidade Federal de Ouro Preto are working on projects that include the following ones:

Proposal for a Solution to Haskell's Multi-parameter Type Class Dilemma The proposal consists of using a simple satisfiability trigger condition: check satisfiability if and only if there exists an unreachable variable in a constraint.

This eliminates the need for functional dependencies (and any other additional mechanism in the language) to tackle ambiguity and overloading resolution.

E-mail messages about the proposal exchanged in Haskell-cafe and Haskell-prime have not been constructive. The discussion in Haskell-cafe deviated to what we see as an ortogonal issue, of import and export of instances (see more about this in the next paragraph).

So unfortunately the proposal has not been incorporated in Haskell yet. A paper about it has been published at SBLP'2009 (see below).

We have implemented the proposal in a proptotype Haskell front-end (https://github.com/rodrigogribeiro/core), and are currently working on this front-end so that it can type all existing Haskell libraries (that use multi-parameter type classes and higher-rank polymorphism).

Controlling the scope of instances in Haskell Marco Gontijo is about to finish his MSc dissertation on the subject. This is a simple and natural change that makes module export and import free of treating instances as a special case. It also allows alternative instances of a class for the same type to be defined and used in different module scopes of a program, eliminates not only problems related to the existence of orphan instances but also the pollution of the global scope by unused instances.

An article about this has been published at SBLP'2011 (see below). Marco Gontijo is currently implementing the proposal, in our Haskell compiler prototype (https://github.com/rodrigogribeiro/core); if time permits, also in GHC.

Decidable type inference for Haskell overloading When types have constraints, decidability of type inference is based mainly on decidability of constraint set satisfiability. We have designed a termination criterion for Haskell's type inference algorithm that deals with all the "complicated cases" (given in e.g. the PPDP'04 and ACM TOPLAS 2005 references below).

A paper about this is being (re)written. An implementation is available at https://github.com/rodrigogribeiro/core.

First Class Overloading and Intersection Types A paper about this has been published at SBLP'2011 (see below).

The work is currently being implemented in our compiler front-end, available at https://github.com/rodrigogribeiro/core.

The Hindley-Milner type system imposes the restriction that function parameters must have monomorphic types. Lifting this restriction and providing system F "first class" polymorphism is clearly desirable, but comes with the difficulty that complete type inference for higher-rank type systems is undecidable. More practical systems supporting higher-rank types have been proposed, which rely on system F, and require appropriate type annotations for the definition of functions with polymorphic type parameters. But these type annotations do inevitably disallow some possible uses of defined higher-rank functions. To avoid this problem, we propose the annotation of intersection types for specifying the types of function parameters used polymorphically inside a function body.

Future work involves extending this work to allow also annotation of union types, supporting then the use (manipulation) of heterogeneous data structures by means of overloaded functions.

Further reading

- *A Solution to Haskell's Multi-parameter Type Class Dilemma*, Carlos Camarão, Rodrigo Ribeiro, Lucília Figueiredo, Cristiano Vasconcellos, SBLP'2009 (13th Brazilian Symposium on Programming Languages). http://www.dcc.ufmg.br/~camarao/CT/solution-to-mptc-dilemma.pdf
- *Controlling the Scope of Instances in Haskell*, Marco Silva, Carlos Camarão, SBLP'2011 (15th Brazilian Symposium on Programming Languages). http://www.dcc.ufmg.br/~camarao/controlling-the-scope-of-instances-in-Haskell-sblp2011.pdf
- *Constraint-set satisfiability for Overloading*, Carlos Camarão, Lucília Figueiredo, Cristiano Vasconcellos, ACM Press Conf. Proceedings of PPDP'04 , 67–77, 2004. http://www.dcc.ufmg.br/~camarao/CT/cs-sat/cssat.pdf
- *A theory of overloading*, Peter J. Stuckey, Martin Sulzmann, ACM TOPLAS 2005, 27(6), 1216–1269. http://portal.acm.org/citation.cfm?id=1108974
- *First Class Overloading via Intersection Type Parameters*, Elton Máximo Cardoso, Carlos Camarão, Lucília Figueiredo, SBLP'2011 (15th

Brazilian Symposium on Programming Languages). http://www.dcc.ufmg.br/~camarao/CT/intersection-type-parameters.pdf

9.4 Artificial Intelligence and Software Technology at Goethe-University Frankfurt

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

Programming language semantics. One of our research topics focuses on programming language semantics, especially on contextual equivalence and bisimilarity. Deterministic call-by-need lambda calculi with letrec provide a semantics for the core language of Haskell. For such an extended lambda calculus we proved correctness of strictness analysis using abstract reduction, and we proved equivalence of the call-by-name and call-by-need semantics, and we proved that applicative bisimilarity is complete w.r.t. contextual equivalence in this calculus.

We also explored several nondeterministic extensions of call-by-need lambda calculi and their applications. A recent result is that for calculi with `letrec` and nondeterminism usual definitions of applicative similarity are unsound w.r.t. contextual equivalence.

We analyzed a higher-order functional language with concurrent threads, monadic IO and synchronizing variables as a core language of Concurrent Haskell. We extended the language by implicit, monadic, and concurrent futures. Using contextual equivalence based on may- and should-convergence, we have shown that several transformations preserve program equivalence, e.g. the monad laws hold in our calculus. An important result is that the language with concurrency conservatively extends the pure core language of Haskell, i.e. all program equivalences for the pure part also hold in the concurrent language. Recently, we introduced a Sestoft-like abstract machine for this language and have shown correctness of the machine.

In a recent research project we try to automate correctness proofs of program transformations. These proofs require to analyze the overlappings between reductions of the operational semantics and transformation steps by computing so-called forking and commuting diagrams. We implemented an algorithm as a combination of several unification algorithms in Haskell which computes these diagrams. Recently, we provided a method to automate the corresponding induction proofs (which use the diagrams) using automated termination provers for term rewriting systems.

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

Further reading

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

9.5 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 — in particular our interest in Erlang has been growing — Haskell provides a major focus and common language for teaching and research.

Our members pursue a variety of Haskell-related projects, some of which are reported in other sections of this report, such as Simon Thompson's text book *Haskell: the craft of functional programming*. Thomas Schilling is writing up his PhD thesis on trace-based dynamic optimisations for Haskell programs. Olaf Chitil developed practial lazy contracts for Haskell using Template Haskell.

We are always looking for PhD students to work with us. We are particularly keen to recruit students interested in programming tools for tracing, refactoring, type checking and any useful feedback for a programmer. The school and university have support for strong candidates: more details at http://www.cs.kent.ac.uk/pg or contact any of us individually by email.

Further reading

- PLAS group: http://www.cs.kent.ac.uk/research/groups/plas/
- Haskell: the craft of functional programming: http://www.haskellcraft.com
- Refactoring Functional Programs: http://www.cs.kent.ac.uk/research/groups/plas/hare.html
- Tracing and debugging with Hat: http://www.haskell.org/hat
- Practial Lazy Typed Contracts for Haskell: http://www.cs.kent.ac.uk/~oc/contracts.html
- Heat: http://www.cs.kent.ac.uk/projects/heat/
- Scion: http://code.google.com/p/scion-lib/

9.6 Formal Methods at DFKI and University Bremen

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

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

We are using the Glasgow Haskell Compiler and many of its extensions to develop the Heterogeneous tool set (Hets). Hets consists of parsers, static analyzers, and proof tools for languages from the CASL family, such as the Common Algebraic Specification Language (CASL) itself (which provides many-sorted first-order logic with partiality, subsorting and induction), HasCASL, CoCASL, CspCASL, and ModalCASL. Other languages supported include Haskell (via Programatica), QBF, Maude, VSE, TPTP, THF, OWL, Common Logic, FPL (logic of functional programs) and LF type theory. The Hets implementation is based on some old Haskell sources such as bindings to uDrawGraph (formerly Davinci) and Tcl/TK that we maintain, but we are moving to a mere web interface based on warp (→ 5.2.2).

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

Further reading

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

9.7 Haskell at Universiteit Gent, Belgium

Report by:	Tom Schrijvers

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:

- As of this academic year, make Haskell part of your curriculum with our brand new *Functional and Logic Programming Languages* course.
- Explore Haskell in depth with one of our Haskell master thesis topics.
- Attend the thriving Ghent Functional Programming Group (→ 9.13).

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

- *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.

- *Monads, Zippers and Views: Virtualizing the Monad Stack*: We make monadic components more reusable and robust to changes by employing two new techniques for *virtualizing* the monad stack: the *monad zipper* and *monad views*. The monad zipper is a higher-order monad transformer that creates virtual monad stacks by ignoring particular layers in a concrete stack. Monad views provide a general framework for monad stack virtualization: they take the monad zipper one step further and integrate it with a wide range of other virtualizations. For instance, particular views allow restricted access to monads in the stack. Furthermore, monad views provide components with a *call-by-reference*-like mechanism for accessing particular layers of the monad stack. With our two new mechanisms, the monadic effects required by components no longer need to be literally reflected in the concrete monad stack. This makes these components more reusable and robust to changes.

This is joint work with Bruno Oliveira, part of which is available together with Mauro Jaskelioff's monad transformer library in the Monatron package on Hackage.

- *EffectiveAdvice:* EffectiveAdvice is a disciplined model of (AOP-style) advice, inspired by Aldrich's Open Modules, that has full support for effects in both base components and advice. EffectiveAdvice is implemented as a Haskell library. Advice is modeled by mixin inheritance and effects are modeled by monads. Interference patterns previously identified in the literature are expressed as combinators. Equivalence of advice, as well as base components, can be checked by equational reasoning. Parametricity, together with the combinators, is used to prove two harmless advice theorems. The result is an effective model of advice that supports effects in both advice and base components, and allows these effects to be separated with strong non-interference guarantees, or merged as needed. This is joint work with Bruno Oliveira and William Cook.

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

9.8 Haskell in Romania

Report by: Dan Popa

This is to report some activities of the Ro/Haskell Group. The Ro/Haskell page becomes more and more known as time goes. Actually, the Ro/Haskell Group is officially a project of the Faculty of Sciences, "V. Alecsandri" Univ. of Bacău, România (http://stiinte.ub.ro) based by volunteers. During the academic year 2011 – 2012 the "Gentle Introduction to Haskell 98" was translated in Romanian and was published by MatrixRom Publishing House (http://www.matrixrom.ro). Romanian title : "O mica introducere in Haskell 98". Prof Paul Hudak had offered a forward for Romanian users.

Website:

On the 7th of Oct. 2011, the main Ro/Haskell's web page counter recorded the total of almost 40 000 times accessed. Some pages was added, included one dedicate to the above book and some pages dedicated to the Leksah IDE. (http://leksah.org)

Books:

The book "The Practice Of Monadic Interpretation" by Dan Popa had been published in November 2008. The book had developed into a full PhD. thesis which was successfully defended in public in September 2010. No English version is available so far.

Actually the Official Publishing House of the Ro/Haskell Group is MatrixRom (www.matrixrom.ro). Speaking of books, the "Gentle introduction to Haskell" was prepared this year and was on the market in a Romanian translation. The introductory chapter (http://www.haskell.org/wikiupload/3/38/Gentle_1-19-v06-3Aprilie.pdf.zip) can be downloaded from http://www.haskell.org/haskellwiki/Gentle where two other versions are available, too: French and of course English.

"An Introduction to Haskell by Examples" is now out of print but if you need, a special pack can be provided based on the agreement of the author ⟨popavdan@yahoo.com⟩. Also available on special request from PIM Publishing House, in Iasi.

Products:

Haskell products like Rodin (a small DSL a bit like C but written in Romanian) begin to spread, proving the power of the Haskell language. The Pseudocode Language Rodin is used as a tool for teaching basics of Computer Science in some high-schools from various cities. Rodin was asked to become a FOSS (Free & Open Source Software) and will be. To have a sort of C using native keywords was a success in teaching basics of Computer Science: algorithms and structured programming.

Linguists:

A group of researchers from the field of linguistics located at the State Univ. from Bacău (The LOGOS Group) is declaring the intention of bridging the gap between semiotics, high level linguistics, structuralism, nonverbal communication, dance semiotics (and some other intercultural subjects) and Computational Linguistics (meaning Pragmatics, Semantics, Syntax, Lexicology, etc.) using Haskell as a tool for real projects. Probably the situation from Romania is not well known: Romania is probably one of those countries where computational linguistics is studied by computer scientists less than linguists. We had begun by publishing an article about The Rodin Project in order to attract linguists. We are trying to extend the base of available books in libraries.

At Bacău "V. Alecsandri" University

We have teaching Haskell at two Faculties: Sciences (The Computers Science being included) and we hope we will work with Haskell with the TI students from the Fac. of Engineering, where a course on Formal Languages was requested.

At Brasov "Transilvania" University

The book "An Introduction to Haskell by Examples" was requested by teachers from the "Transilvania" Univ. of Brasov., where a master course on functional programming in Haskell was introduced.

Notions:

We are promoting new notions: pseudoconstructors over monadic values (which act both as semantic representations and syntactic structure), modular trees (expanding trees beyound the fixity of the data declarations) and ADFA — adaptive/adaptable determinist finite automata. A dictionary of new notions and concepts is not made, making difficult to launch new ideas and also to track work of the authors.

Unsolved problems:

PhD. advisors (specialized in monads, language engineering, and Haskell) are almost impossible to find. This fact seems to block somehow the hiring of good specialists in Haskell. Also it is difficult to track the Haskell related activity from various universities, like those from: Sibiu, Baia Mare, Timisoara. Please report them using the below address.

Contact

⟨popavdan@yahoo.com⟩

Further reading

- Ro/Haskell: http://www.haskell.org/haskellwiki/Ro/Haskell
- Rodin: http://www.haskell.org/haskellwiki/Rodin
- Gentle introduction to Haskell (Ro): http://www.haskell.org/haskellwiki/Gentle
- ADFA: http://www.haskell.org/haskellwiki/ADFA
- Report from: http://stiinte.ub.ro (the Faculty I belong to)

9.9 fp-syd: Functional Programming in Sydney, Australia

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

We are a seminar and social group for people in Sydney, Australia, interested in Functional Programming and related fields. Members of the group include users of Haskell, Ocaml, LISP, Scala, F#, Scheme and others. We have 10 meetings per year (Feb–Nov) and meet on the third Thursday of each month. We regularly get 20–30 attendees, with a 70/30 industry/research split. Talks this year have included material on compilers, theorem proving, type systems, Template Haskell and a couple of different Haskell libraries. 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/

9.10 Functional Programming at Chalmers

Report by:	Jean-Philippe Bernardy

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

Property-based testing QuickCheck, developed at Chalmers, is one of the standard tools for testing Haskell programs. We are currently applying the QuickCheck approach to Erlang software, together with Ericsson, Quviq, and others. The venerable QuickCheck tool is nowadays complemented with PULSE, the ProTest User-Level Scheduler for Erlang, which has been used to find race conditions in industrial software. We have also shown how to sucessfully apply QuickCheck to polymorphic properties: http://publications.lib.chalmers.se/cpl/record/index.xsql?pubid=99387.

Natural language technology Grammatical Framework (http://www.haskell.org/communities/11-2010/html/report.html#sect9.7.3) is a declarative language for describing natural language grammars. It is useful in various applications ranging from natural language generation, parsing and translation to software localization. The framework provides a library of large coverage grammars for currently fifteen languages from which the developers could derive smaller grammars specific for the semantics of a particular application.

Parser generator and template-haskell BNFC-meta is an embedded parser generator, presented at the Haskell Symposium 2011. Like the BNF Converter, it generates a compiler front end in Haskell. Two aspects distinguish BNFC-meta from BNFC and other parser generators:

- BNFC-meta is not a program but a library (the parser description is embedded in a quasi-quote).
- BNFC-meta automatically provides quasi-quotes for the specified language. This includes a powerful and flexible facility for anti-quotation.

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

Generic Programming Starting with Polytypic Programming in 1995 there is a long history of generic programming research at Chalmers. Recent developments include fundamental work on "Proofs for Free" (extensions of the parametricity & dependent types work from ICFP 2010, now published in JFP 2012) and a Haskell Symposium paper on "Embedded Parser Generators" (see BNFC-meta above). Patrik Jansson leads a work-package on DSLs within the EU project "Global Systems Dynamics and Policy" (http://www.gsdp.eu/, started Oct. 2010). If you want to apply DSLs, Haskell, and Agda to help modelling Global Systems Science, please get in touch! Currently Johan Jeuring from Utrecht is visiting us on sabbatical. Jansson and Bernardy have also just started a new project called "Strongly Typed Libraries for Programs and Proofs".

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

Type theory Type theory is strongly connected to functional programming research. Many dependently-typed programming languages and type-based proof assistants have been developed at Chalmers. The Agda system (→ 4.1) is the latest in this line, and is of particular interest to Haskell programmers. We encourage you to experiment with programs and proofs in Agda as a "dependently typed Haskell".

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

- **Feldspar** (→ 7.15.3) is a domain-specific language for digital signal processing (DSP), developed in co-operation by Ericsson, Chalmers FP group and Eötvös Loránd (ELTE) University in Budapest.
- **Obsidian** is a language for data-parallel programming targeting GPGPUs.

The following languages are not actively developed at the moment:

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

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

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

theory as lemmas, to aid proving later properties which may require them. For more information, see the draft paper http://web.student.chalmers.se/~danr/hipspec-atx.pdf

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

We are currently teaching a new MSc level course on "Parallel Functional Programming", which already featured prominent Haskellers as guest lecturers, such as Andres Löh and Simon Marlow.

9.11 Functional Programming at KU

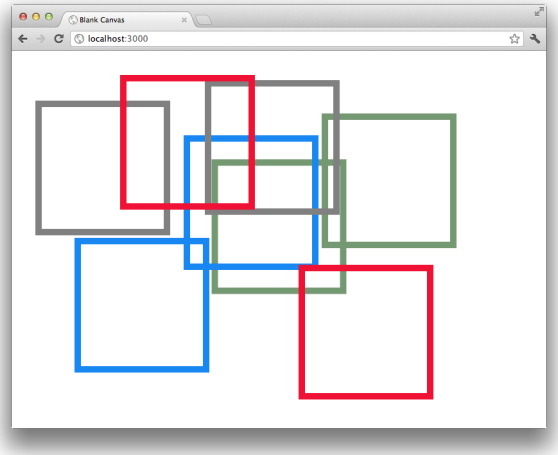
Report by:	Andy Gill
Status:	ongoing

Functional Programming is vibrant at KU and the Computer Systems Design Laboratory in ITTC! The System Level Design Group (lead by Perry Alexander) and the Functional Programming Group (lead by Andy Gill) together form the core functional programming initiative at KU. Apart from Kansas Lava (→ 7.13.2) and HERMIT (→ 7.5.1), there are several other FP and Haskell related things going on, primarily in the area of web technologies.

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

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

On top of `Scotty`, we have built a simple interface into the HTML5 Canvas mechanism, called `blank-canvas`. This was constructed primarily as a teaching tool and a proof-of-concept design. Here is an example of a teaching application which prints squares to the canvas, based on where the user clicks the mouse.



Simple interactive games can be developed using this API, and new Haskell programmers found it straightforward to use. Unbeknown to us, `blank-canvas` was also used in a high-school level functional programming mentoring effort being lead by Alwyn Goodloe.

We are now working to generalize the ideas in `blank-canvas` by creating a framework to handle arbitrary asynchronous Javascript events and DOM manipulation via a server-side Haskell DSL. This DSL will make it possible to control a browser-based UI using well known Haskell patterns, such as Functional Reactive Programming.

All packages are available from hackage, or will be shortly.

Further reading

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

9.12 San Simón Haskell Community

Report by:	Antonio Mamani
Participants:	Carlos Gomez

The San Simón Haskell Community from San Simón University Cochabamba-Bolivia, is an informal Spanish

group that aims to learn, share information, knowledge and experience related to the functional paradigm.

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

Projects in the 2nd Open House Haskell Community:

1. **L-System** — Application that renders an L-System using wxHaskell (http://hackage.haskell.org/package/lsystem) [Carlos Gomez]
2. **Compiler IDL - Java** — Generate code from IDL to Java. [Richard Jaldin]
3. **Mini Java** — Mini-Java compiler from scratch. [Antonio Mamani]
4. **3S Functional Web Browser** — Bachelor theses project about experimenting the implementation of a web browser with Haskell. (http://hsbrowser.wordpress.com/3s-functional-web-browser/) [Carlos Gomez]

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

9.13 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 years, holding meetings on a regular basis.

We have reported in previous HCARs on the first nine meetings. Since November 2011, we had two meetings. The GhentFPG #10 meeting in December 2011 involved a problem solving session, where participants tackled two problems and solutions were presented and discussed at the end of the meeting.

In the GhentFPG #11 March 2012 meeting, we had two talks:

- Jasper Van der Jeugt — Digestive Functors
- Jeroen Janssen — Presentation on the Typeclassopedia paper

The attendance at the meetings usually varies between 10 to 15 people, with significantly less attendance for problem solving activities.

At this point, we have plans for organising another hackathon, somewhere in the fall of 2012.

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