Haskell Communities and Activities Report

http://www.haskell.org/communities/

Fourteenth Edition — May, 2008

Andres Löh, Janis Voigtländer (eds.)

Andy Adams-Moran
Tiago Miguel Laureano Alves
Sengan Baring-Gould
Clifford Beshers
Chris Brown
Andrew Butterfield
Sterling Clover
Jason Dagit
Henrique Ferreiro García
Leif Frenzel
Daniel Gorin
Bastiaan Heeren
Christopher Lane Hinson
Antti-Juhani Kaijanaho
Edward Kmett
Eric Kow
Salvador Lucas
Conor McBride
Blažević Mario
Arie Middelkoop
Rishiyur Nikhil
Claus Reinke
David Sabel
Paulo Silva
Ganesh Sittampalam
Don Stewart
Martin Sulzmann
Henning Thielemann
Andrea Vezzosi
Edsko de Vries
Eelis van der Weegen

alpheccar
Apfelmus
Alistair Bayley
Joachim Breitner
Bjorn Buckwalter
Olaf Chitil
Duncan Coutts
Robert Dockins
Sebastian Fischer
Richard A. Frost
Murray Gross
Wolfgang Jeltsch
Graham Hutton
Oleg Kiselyov
Lennart Kolmodin
Andres Löh
Ian Lynagh
Neil Mitchell
Simon Marlow
Matthew Naylor
Bryan O'Sullivan
Alberto Ruiz
Matthew Sackman
Axel Simon
Jim Snow
Jon Strait
Wouter Swierstra
Peter Thiemann
Miguel Vilaca
David Waern
Ashley Yakeley
Bulat Ziganshin

Lloyd Allison
Carlos Areces
Jean-Philippe Bernardy
Niklas Broberg
Denis Bueno
Jan Christiansen
Nils Anders Danielsson
Keith Fahlgren
Simon Frankau
George Giorgidze
Jurriaan Hage
Kevin Hammond
Wolfram Kahl
Dirk Kleeblatt
Slawomir Kolodynski
Rita Loogen
Ketil Malde
Christian Maeder
Steffen Mazanek
Jürgen Nicklisch-Franken
Simon Peyton Jones
Colin Runciman
Uwe Schmidt
Ben Sinclair
Dominic Steinitz
Jennifer Streb
Hans van Thiel
Phil Trinder
Janis Voigtländer
Malcolm Wallace
Brent Yorgey

Preface

This is the 14th edition of the Haskell Communities and Activities Report. There has been a transition in editorship which went very smoothly, also thanks to the many responsive contributors who where as helpful to the new editor as they have been to Andres during the last years.

As usual, entries that are completely new (or have been revived after having disappeared temporarily) are formatted using a blue background. Updated entries have a header with a blue background. In most cases of entries that have not been changed for a year or longer, these have been dropped.

The report has been somewhat restructured, so if you do not find your favourite entry at once: please check the table of contents again; maybe the entry is just elsewhere than where you last saw it. If for some entry you think a different place in the report would be more appropriate, please give sign for next time. Also, to simplify organisation, only one author is now assigned to every entry. Where previously several authors were given, the entry has been assigned to the one sending it in, and any further given authors have been added as participants. Of course, authorship can be reassigned with the next edition.

By now, the report has reached a considerable size. This does not only have to do with the pleasantly high number of entries contained, but also with the fact that many of them are growing "through accumulation". To counter this a bit, and encourage focusing on describing the most recent activities, the next edition of HCAR will have a (liberal) length limit on entries, except for a few projects of central importance (the compilers, Cabal and Hackage, ...). More details around November — watch the mailing lists for announcements.

But now enjoy the report and see what other Haskellers have been up to lately. Any kind of feedback is of course very welcome ⟨hcar@haskell.org⟩.

Andres Löh, Universiteit Utrecht, The Netherlands
Janis Voigtländer, Technische Universität Dresden, Germany

Contents

1 General

1.1 HaskellWiki and haskell.org

Report by:	Ashley Yakeley
Participants:	John Peterson, Olaf Chitil

HaskellWiki is a MediaWiki installation running on haskell.org, including the haskell.org "front page". Anyone can create an account and edit and create pages. Examples of content include:

- Documentation of the language and libraries
- Explanation of common idioms
- Suggestions and proposals for improvement of the language and libraries
- Description of Haskell-related projects
- News and notices of upcoming events

We encourage people to create pages to describe and advertise their own Haskell projects, as well as add to and improve the existing content. All content is submitted and available under a "simple permissive" license (except for a few legacy pages).

In addition to HaskellWiki, the haskell.org website hosts some ordinary HTTP directories. The machine also hosts mailing lists. There is plenty of space and processing power for just about anything that people would want to do there: if you have an idea for which HaskellWiki is insufficient, contact the maintainers, John Peterson and Olaf Chitil, to get access to this machine.

Further reading

- http://haskell.org/
- http://haskell.org/haskellwiki/Mailing_Lists

1.2 `#haskell`

Report by:	Don Stewart

The `#haskell` IRC channel is a real-time text chat where anyone can join to discuss Haskell. The channel has continued to grow in the last six months, now averaging around 420 users, with a record 479 users (up from 436 six months ago). It is one of the largest channels on freenode. The irc channel is home to hpaste and lambdabot (→ 6.8.1), two useful Haskell bots. Point your IRC client to irc.freenode.net and join the `#haskell` conversation!

For non-English conversations about Haskell there are now:

- `#haskell.de` – German speakers
- `#haskell.dut` – Dutch speakers
- `#haskell.es` – Spanish speakers
- `#haskell.fi` – Finnish speakers
- `#haskell.fr` – French speakers
- `#haskell.hr` – Croatian speakers
- `#haskell.it` – Italian speakers
- `#haskell.jp` – Japanese speakers
- `#haskell.no` – Norwegian speakers
- `#haskell_ru` – Russian speakers
- `#haskell.se` – Swedish speakers

Related Haskell channels are now emerging, including:

- `#haskell-overflow` – Overflow conversations
- `#haskell-blah` – Haskell people talking about anything except Haskell itself
- `#gentoo-haskell` – Gentoo/Linux specific Haskell conversations (→ 2.6.1)
- `#haskell-books` – Authors organising the collaborative writing of the Haskell Wikibook (→ 1.6.3)
- `#darcs` – Darcs revision control channel (→ 6.1.1)
- `#ghc` – GHC developer discussion (→ 2.1)
- `#happs` – HAppS Haskell Application Server channel
- `#xmonad` – XMonad, a tiling window manager (→ 6.1.2)

Further reading

http://haskell.org/haskellwiki/IRC_channel

1.3 The Monad.Reader

Report by:	Wouter Swierstra

There are plenty of academic papers about Haskell and plenty of informative pages on the HaskellWiki (→ 1.1). 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 maybe do 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, I have moved a lot of old articles from the old MoinMoin wiki to the new MediaWiki wiki. Unfortunately, I do not have the time to reformat all the old articles. If you fancy a go at tidying an article or two, I would really appreciate your help!

I am always interested in new submissions, whether you are an established researcher or fledgling Haskell programmer. Check out the Monad.Reader homepage for all the information you need to start writing your article.

Further reading

http://www.haskell.org/haskellwiki/The_Monad.Reader

1.4 Haskell Weekly News

Report by: Don Stewart

The Haskell Weekly News (HWN) is an irregular newsletter covering developments in Haskell. Content includes announcements of new projects, jobs, discussions from the various Haskell communities, notable project commit messages, Haskell in the blogspace, and more. The Haskell Weekly News also publishes latest releases uploaded to Hackage.

It is published in html form on The Haskell Sequence, via mail on the Haskell mailing list, on Planet Haskell (→ 1.5), and via RSS. Headlines are published on haskell.org (→ 1.1).

Further reading

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

1.5 Planet Haskell

Report by: Antti-Juhani Kaijanaho

Planet Haskell is an aggregator of Haskell people's blogs and other Haskell-related news sites. As of April 2008 content from 92 blogs and other sites is being republished in a common format.

A common misunderstanding about Planet Haskell is that it republishes only Haskell content. That is not its mission. A Planet shows what is happening in the community, what people are thinking about or doing. Thus Planets tend to contain a fair bit of "off-topic" material. Think of it as a feature, not a bug.

For information on how to get added to Planet, please read http://planet.haskell.org/policy.html.

Further reading

http://planet.haskell.org/

1.6 Books and tutorials

1.6.1 Programming in Haskell

Report by: Graham Hutton

Haskell is one of the leading languages for teaching functional programming, enabling students to write simpler and cleaner code, and to learn how to structure and reason about programs. This introduction is ideal for beginners: it requires no previous programming experience and all concepts are explained from first principles via carefully chosen examples. Each chapter includes exercises that range from the straightforward to extended projects, plus suggestions for further reading on more advanced topics. The presentation is clear and simple, and benefits from having been refined and class-tested over several years.

Features include: freely accessible powerpoint slides for each chapter; solutions to exercises, and examination questions (with solutions) available to instructors; downloadable code that is fully compliant with the latest Haskell release.

Publication details:

- Published by Cambridge University Press, 2007. Paperback: ISBN 0521692695; Hardback: ISBN: 0521871727; eBook: ISBN 051129218X.

In-depth review:

- Duncan Coutts, The Monad.Reader (→ 1.3), http://www.haskell.org/sitewiki/images/0/03/TMR-Issue7.pdf

Further reading

http://www.cs.nott.ac.uk/~gmh/book.html

1.6.2 Real World Haskell

Report by: Bryan O'Sullivan
Participants: John Goerzen, Don Stewart
Status: active development

We are working on a book, "Real World Haskell", about the practical application of Haskell to everyday programming problems. The book will be published in the second half of 2008 by O'Reilly.

Our intended audience is programmers with no background in functional languages. We explore a diverse set of topics, among which are the following.

- Basics of Haskell and functional programming
- Developing software using standard tools like GHC and the Cabal packaging system
- Code coverage, quality assurance, and performance analysis
- Putting theory to work: working with and creating monoids, normal and applicative functors, monads, and monad transformers

- Applied topics: databases, filesystems, GUI programming, web and other network clients, web servers
- Concurrent, parallel, and transactional programming
- Error handling in pure and impure code
- Interfacing to C libraries
- Many case studies and runnable code examples

At the time of writing (late April, 2008) we have first drafts of about 75 per cent of the book written. We expect it to come to about 30 chapters in total.

We are excited to be publishing the book under a Creative Commons License. As we write chapters, we publish them online for people to read and comment on, and we incorporate feedback from our readers when we rewrite drafts of chapters. The level of community interest has been remarkable: so far, we have received over 4,000 comments on the chapters we have made available.

Further reading

- Book site: http://book.realworldhaskell.org/
- Blog with progress and publication updates: http://www.realworldhaskell.org/blog/

1.6.3 Haskell Wikibook

Report by:	Apfelmus
Participants:	Eric Kow, David House, Joeri van Eekelen, and other contributors
Status:	active development

The goal of the Haskell wikibook project is to build a community textbook about Haskell that is at once free (as in freedom and in beer), gentle, and comprehensive. We think that the many marvellous ideas of lazy functional programming can and thus should be accessible to everyone in a central place.

Since the last report, the wikibook has been advancing rather slowly. The rewrite of the Monad chapters is still in progress and material about lazy evaluation is still being written. Of course, additional authors and contributors that help writing new contents or simply spot mistakes and ask those questions we had never thought of are more than welcome!

Further reading

- http://en.wikibooks.org/wiki/Haskell
- Mailing list: ⟨wikibook@haskell.org⟩

1.6.4 Gtk2Hs tutorial

Report by:	Hans van Thiel

Most of the original GTK+2.0 tutorial by Tony Gail and Ian Main has been adapted to Gtk2Hs (→ 5.8.1), which is the Haskell binding to the GTK GUI library.

The Gtk2Hs tutorial also builds on "Programming with gtkmm" by Murray Cumming et al. and the Inti (Integrated Foundation Classes) tutorial by the Inti team.

The Gtk2Hs tutorial assumes intermediate level Haskell programming skills, but no prior GUI programming experience.

It has been translated into Spanish, by Laszlo Keuschnig, and both versions are available on Haskell darcs.

See: http://darcs.haskell.org/gtk2hs/docs/tutorial/Tutorial_Port/

1. Introduction
2. Getting Started
3. Packing
   - 3.1 Packing Widgets
   - 3.2 Packing Demonstration Program
   - 3.3 Packing Using Tables
4. Miscellaneous Widgets
   - 4.1 The Button Widget
   - 4.2 Adjustments, Scale, and Range
   - 4.3 Labels
   - 4.4 Arrows and Tooltips
   - 4.5 Dialogs, Stock Items, and Progress Bars
   - 4.6 Text Entries and Status Bars
   - 4.7 Spin Buttons
5. Aggregated Widgets
   - 5.1 Calendar
   - 5.2 File Selection
   - 5.3 Font and Colour Selection
   - 5.4 Notebook
6. Supporting Widgets
   - 6.1 Scrolled Windows
   - 6.2 EventBoxes and ButtonBoxes
   - 6.3 The Layout Container
   - 6.4 Paned Windows and Aspect Frames
7. Action Based Widgets
   - 7.1 Menus and Toolbars
   - 7.2 Popup Menus, Radio Actions, and Toggle Actions

Appendix: Drawing with Cairo: Getting Started

The Glade tutorial, an introduction to visual Gtk2Hs programming, has been updated to Glade 3 by Alex Tarkovsky. It is available on: http://haskell.org/gtk2hs/docs/tutorial/glade/ This tutorial has also been translated into Spanish, by Laszlo Keuschnig, but it is currently only available on: http://home.telfort.nl/sp969709/glade/es-index.html

1.6.5 Oleg's Mini tutorials and assorted small projects

Report by: Oleg Kiselyov

The collection of various Haskell mini tutorials and assorted small projects (http://okmij.org/ftp/Haskell/) has received three additions:

Binary type arithmetic

With Chung-chieh Shan we introduce a type-level Haskell library for arbitrary precision binary arithmetic over natural *kinds*. The numerals are specified in the familiar big-endian bit notation. The library supports addition/subtraction, predecessor/successor, multiplication/division, exp2, all comparisons, GCD, and the maximum. At the core of the library are multi-mode ternary *relations* `Add` and `Mul` where *any* two arguments determine the third. Such relations are especially suitable for specifying static arithmetic constraints on computations. The type-level numerals have no run-time representation; correspondingly, all arithmetic operations are done at compile time and have no effect on run-time.

Two applications of the library for safe system programming are described in the next item.

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

Typed memory areas and time-parameterised monads, for safe embedded and systems programming

We argue that Haskell as supported by GHC today can be used for safe system programming, statically assuring safe handling of raw memory pointers, device registers, and other low-level resources. Safety is assured by the type system and has no run-time overhead. We demonstrate two extensive examples: (i) one built around raw pointers, to track and arbitrate the size, alignment, write permission, and other properties of memory areas in the presence of indexing, casting, and iteration; (ii) the other built around a device register, to enforce protocol and timing requirements while reading from the register.

We also demonstrate custom kinds and predicates; type-level numbers, functions, and records; and mixed type- and term-level programming.

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

Polymorphic variants: solving the expression problem

There have been several proposals for Haskell extensions to support open polymorphic variants, i.e., extensible recursive open sum datatypes similar to polymorphic variants of OCaml. We demonstrate that Haskell as it is — the HList library (→ 5.5.5) — already supports polymorphic variants, with automatic variant subtyping. HList thus solves the familiar (in OO, at least) "expression problem" — the ability to add new alternatives to a datatype and extend old processing functions to deal with the extended variant, maximally reusing old code without changing it.

Our polymorphic variants are literally open coproducts: *dual* of, literally *negated* extensible polymorphic records of HList. Our encoding of sums is the straightforward Curry-Howard image of the DeMorgan law of the negation of disjunction.

Our implementation of polymorphic variants in terms of HList records uses no type classes, no type-level programming or any other type hacking. In fact, there are no type annotations, type declarations or any other mentioning of types, except in the comments. The code is included in the HList library.

http://okmij.org/ftp/Haskell/generics.html#PolyVariant

2 Implementations

2.1 The Glasgow Haskell Compiler

Report by:	Simon Peyton Jones
Participants:	Tim Chevalier, Aaron Tomb, Roman Leshchinskiy, Gabrielle Keller, Max Bolingbroke, John Dias, Thomas Schilling, and many others

The last six months have been a time of consolidation for GHC. We have done many of the things described in the last HCAR, but there are few new headline items to report, so this status report is briefer than usual.

Highlights of the last six months

- *Several simple language extensions* are now solidly in the HEAD.
  - Record syntax: wild-card patterns, punning, and field disambiguation
  - View patterns
  - Generalised list comprehensions
  - Quasi-quoting
- *Type-indexed families.* We learned a lot by writing a paper about the question of type inference in the presence of type families (and existentials, and GADTs): http://research.microsoft.com/~simonpj/papers/assoc-types. The implementation has not quite caught up with the paper and is still incomplete in many ways, but it is a focus of active work and already usable. If you are interested in type families, now would be a good time to grab a development snapshot of GHC, write some programs or port your favourite program using functional dependencies, and then, let us know what does and what does not work for you.
- *Parallel garbage collection.* Much implementation work, and a paper for ISMM 2008: http://research.microsoft.com/~simonpj/papers/parallel-gc/index.htm.
- *Impredicative polymorphism.* We are not happy with GHC's current implementation of impredicative polymorphism, which is rather complicated and ad hoc. Dimitrios (with Simon and Stephanie) wrote a paper about a new and better approach: http://research.microsoft.com/~simonpj/papers/boxy. At the same time, Daan Leijen has been working on his closely-related design: http://research.microsoft.com/users/daan/pubs.html. Daan's design has a much simpler implementation, in exchange for an (arguably) less-predictable specification. Which of these two should we implement? Let us know!
- *External Core.* Tim Chevalier has updated the External Core format to incorporate type equality coercions and other recent GHC changes, as well as extending the stand-alone External Core tools (a parser, typechecker, and interpreter that can be built separately from GHC) to handle this new format. As of now, it is only possible to use GHC's front-end to pipe External Core into other back-end tools — GHC still cannot read in External Core that was produced by other tools (or itself). But this is an improvement over the bit-rotted state into which External Core had fallen. Aaron Tomb contributed much to this effort as well.

Nested data parallelism

We have been working hard on Data Parallel Haskell, especially Roman Leshchinskiy and Gabriele Keller. It has turned out be be hard to get the entire transformation and optimisation stack to work smoothly, and we have not made progress announcements because we do not want to yell about it until it Actually Works. But it is the biggest single GHC focus: Roman works on it full time.

Large parts of the major pieces are in place. GHC contains a shiny new vectoriser that turns scalar into data-parallel functions. Moreover, the sequential and parallel array libraries targeted by the vectoriser have been steadily growing. We managed to successfully run small applications, such as an n-body simulator based on the Barnes-Hut algorithm, but the vectoriser and library are still awkward to use and need to be more robust before being useful to a wider audience. We also need to improve performance.

We expect to release a working version of Data Parallel Haskell as part of GHC 6.10 (see below).

Other current activities

- Max Bolingbroke resurrected the *static argument transformation.* It does not matter for most programs, but has a big effect on a few.
- Work on the *back end* has been stalled, but John Dias started a 6-month internship in April, so expect progress on this front.
- Thomas Schilling is doing a Google Summer of Code project to improve the *GHC API.*

- Max Bolingbroke is doing a Google Summer of Code project to make it easy to build a *plug-in* for GHC; for example, a new optimisation or analysis pass.

Release plans

We plan to release GHC 6.8.3 at the end of May 2008, with many bug-fixes but no new features.

We plan to release GHC 6.10 around the time of ICFP, with significant new features. The up-to-date list of new stuff is kept at http://hackage.haskell.org/trac/ghc/wiki/Status/Releases, but here's a quick summary:

- Simple language extensions (mentioned above)
- Type-indexed families
- Data Parallel Haskell
- Parallel garbage collection
- Extensible exceptions
- External Core
- Shared libraries
- Improved back end
- Further library reorganisation

2.2 nhc98

Report by:	Malcolm Wallace
Status:	stable, maintained

nhc98 is a small, easy to install, compiler for Haskell'98. nhc98 is still very much alive and working, although it does not see much new development these days. The last public release (1.20) was in November 2007, for compatibility with ghc-6.8.x. Continuing maintenance ensures that common library packages build in their most recent versions.

Further reading

- http://haskell.org/nhc98
- `darcs get` http://darcs.haskell.org/nhc98

2.3 yhc

Report by:	Neil Mitchell
Participants:	Dimitry Golubovsky

The York Haskell Compiler (yhc) is a fork of the nhc98 compiler (→ 2.2), with goals such as increased portability, platform independent bytecode, integrated Hat (→ 4.3.5) support, and generally being a cleaner code base to work with. Yhc now compiles and runs almost all Haskell 98 programs, has basic FFI support — the main thing missing is haskell.org base libraries, which is being worked on.

Since the last HCAR there have been a number of projects making use of the Yhc.Core library. Of particular interest is the Javascript backend, which has reached its first milestone. An experimental Yhc web service has been launched, to allow people to experiment with the Javascript backend.

Further reading

- Homepage: http://www.haskell.org/haskellwiki/Yhc
- Darcs repository: http://darcs.haskell.org/yhc
- Yhc Javascript Web Service http://www.haskell.org/haskellwiki/Yhc_web_service

2.4 The Helium compiler

Report by:	Jurriaan Hage
Participants:	Bastiaan Heeren, Arie Middelkoop

Helium is a compiler that supports a substantial subset of Haskell 98 (but, e.g., n+k patterns are missing). Type classes are restricted to a number of built-in type classes and all instances are derived. The advantage of Helium is that it generates novice friendly error feedback. The latest versions of the Helium compiler are available for download from the new website located at http://www.cs.uu.nl/wiki/Helium. This website also explains in detail what Helium is about, what it offers, and what we plan to do in the near and far future.

We are still working on making version 1.7 available, mainly a matter of updating the documentation and testing the system. Internally little has changed, but the interface to the system has been standardised, and the functionality of the interpreters has been improved and made consistent. We have made new options available (such as those that govern where programs are logged to). The use of Helium from the interpreters is now governed by a configuration file, which makes the use of Helium from the interpreters quite transparent for the programmer. It is also possible to use different versions of Helium side by side (motivated by the development of Neon (→ 5.2.7)).

2.5 The Reduceron

Report by:	Matthew Naylor
Participants:	Colin Runciman, Neil Mitchell
Status:	Experimental

The Reduceron is a prototype of a special-purpose graph reduction machine, built using an FPGA. It can access up to eight graph nodes in parallel on each of its stack, heap, and combinator memories. The goal so far has been to optimise function unfolding. Eight combinator nodes can be instantiated with eight stack elements and placed on the heap, all in a single cycle.

The Reduceron is a simple machine, containing just four instructions and a garbage collector, and executes core Haskell almost directly. The translator to bytecode and the FPGA machine are both implemented in Haskell, the latter using Lava.

See the URL below for details and results. Since the last HCAR, I have written a thesis chapter about it, with all the gory details unveiled! Further experiments are planned.

Further reading

http://www.cs.york.ac.uk/~mfn/reduceron2/

2.6 Platforms

2.6.1 Haskell in Gentoo Linux

Report by:	Lennart Kolmodin

GHC version 6.8.2 has been in Gentoo since late last year, and is about to go stable. All of the 60+ Haskell libraries and tools work with it, too. There are also GHC binaries available for alpha, amd64, hppa, ia64, sparc, and x86.

Browse the packages in portage 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 a Haskell overlay providing another 200 packages. Thanks to the recent progress of Cabal and Hackage (→ 5.1), we have written a tool called "hackport" (initiated by Henning Günther) to generate Gentoo packages that rarely need much tweaking.

The overlay is available at http://haskell.org/haskellwiki/Gentoo. Using Darcs (→ 6.1.1), it is easy to keep updated and send patches. It is also available via the Gentoo overlay manager "layman". If you choose to use the overlay, then problems should be reported on IRC (`#gentoo-haskell` on freenode), where we coordinate development, or via email ⟨haskell@gentoo.org⟩.

Lately a few of our developers have shifted focus, and only a few developers remain. If you would like to help, which would include working on the Gentoo Haskell framework, hacking on hackport, writing ebuilds, and supporting users, please contact us on IRC or email as noted above.

2.6.2 OpenBSD Haskell

Report by:	Don Stewart
Participants:	Matthias Kilian

Haskell support on OpenBSD is now taken over by Matthias Kilian, who has updated GHC and related tools for this platform. OpenBSD support for the GHC head branch continues.

Further reading

http://ports.openbsd.nu/lang/ghc/

3 Language

3.1 Extensions of Haskell

3.1.1 Haskell Server Pages (HSP)

Report by: Niklas Broberg
Status: active development

Haskell Server Pages (HSP) is an extension of Haskell targeted at writing dynamic web pages. Key features and selling points include:

- Use literal XML syntax in your Haskell code for creating values of appropriate datatypes. (Note though that writing literal XML is quite optional, if you, like me, do not really enjoy that language.)
- Guarantees that XML output is well-formed (and an HTML output mode if that is what you need).
- A model that gives easy access to necessary environment variables.
- Simple programming model that is easy to use even for non-experienced Haskell programmers, in particular with a very simple transition from static XML pages to dynamic HSP pages.
- Easy integration with a DSL called HJScript that makes it easy to write client-side (JavaScript) scripts.
- An extension of HAppS that can serve HSP pages on the fly, making deployment of pages really simple.

After a few years in hiatus, we have recently picked up development of HSP again and intend to continue developing and supporting it. The latest full release of HSP was in March 2008, and introduced the easy integration between server-side and client-side code. A lot of work has gone into the development version since then, and a new release will probably take place before this HCAR report is published.

HSP is and will be continuously released onto Hackage. It consists of a series of interdependent packages with package hsp as the main top-level starting point, and package happs-hsp for integration with HAppS. The best way to keep up with development is to grab the darcs repositories, all located under http://code.haskell.org/HSP.

Further reading

http://haskell.org/haskellwiki/HSP

3.1.2 GpH — Glasgow Parallel Haskell

Report by: Phil Trinder
Participants: Abyd Al Zain, Mustafa Aswad, Jost Berthold, Murray Gross, Kevin Hammond, Vladimir Janjic, Hans-Wolfgang Loidl, Greg Michaelson

Status

A complete, GHC-based implementation of the parallel Haskell extension GpH and of evaluation strategies is available. Extensions of the runtime-system and language to improve performance and support new platforms are under development.

System Evaluation and Enhancement

- A major revision of the parallel runtime environment for GHC 6.8 is currently under development. The GpH and Eden (→ 3.1.3) parallel Haskells share much of the implementation technology and both are being used for parallel language research and in the SCIEnce project (see below).
- We are exploring the use of GpH on multicore architectures.
- We are teaching parallelism to undergraduates using GpH at Heriot-Watt and Philipps-Universität Marburg.

GpH Applications

- As part of the SCIEnce EU FP6 I3 project (026133) (→ 8.7) (April 2006 - April 2011) we use GpH and Eden as middleware to provide access to computational grids from Computer Algebra (CA) systems, including GAP, Maple MuPad, and KANT. We have designed, implemented, and are evaluating the SymGrid-Par interface that facilitates the orchestration of computational algebra components into high-performance parallel applications.

  In recent work we have demonstrated that SymGrid-Par is capable of exploiting a variety of modern parallel/multicore architectures without any change to the underlying CA components; and that SymGrid-Par is capable of orchestrating heterogeneous computations across a high-performance computational Grid.

Implementations

The GUM implementation of GpH is available in two main development branches.

- The focus of the development has switched to versions tracking GHC releases, currently GHC 6.8, and the development version is available upon request to the GpH mailing list (see the GpH web site).
- The stable branch (GUM-4.06, based on GHC-4.06) is available for RedHat-based Linux machines. The stable branch is available from the GHC CVS repository via tag gum-4-06.

We are exploring new, *prescient* scheduling mechanisms for GpH.

Our main hardware platforms are Intel-based Beowulf clusters and multicores. Work on ports to other architectures is also moving on (and available on request):

- A port to a Mosix cluster has been built in the Metis project (→ 8.6) at Brooklyn College, with a first version available on request from Murray Gross.

Further reading

- GpH homepage: http://www.macs.hw.ac.uk/~dsg/gph/
- Stable branch binary snapshot: ftp://ftp.macs.hw.ac.uk/pub/gph/gum-4.06-snap-i386-unknown-linux.tar
- Stable branch installation instructions: ftp://ftp.macs.hw.ac.uk/pub/gph/README.GUM

Contact

⟨gph@macs.hw.ac.uk⟩, ⟨mgross@dorsai.org⟩

3.1.3 Eden

Report by: Rita Loogen

Description

Eden has been jointly developed by two groups at Philipps-Universität Marburg, Germany, and Universidad Complutense de Madrid, Spain. The project has been ongoing since 1996. Currently, the team consists of the following people:

in Madrid: Ricardo Peña, Yolanda Ortega-Mallén, Mercedes Hidalgo, Fernando Rubio, Alberto de la Encina, Lidia Sánchez-Gil

in Marburg: Rita Loogen, Jost Berthold, Mischa Dieterle, Oleg Lobachev

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), synchronisation, and process handling.

Eden's main constructs are process abstractions and process instantiations. The function `process :: (a -> b) -> Process a b` embeds a function of type `(a -> b)` into a *process abstraction* of type `Process a b` which, when instantiated, will be executed in parallel. *Process instantiation* is expressed by the predefined infix operator `( # ) :: Process a b -> a -> b`. Higher-level coordination is achieved by defining *skeletons*, ranging from a simple parallel map to sophisticated replicated-worker schemes. They have been used to parallelise a set of non-trivial benchmark programs.

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.

Implementation

A major revision of the parallel Eden runtime environment for GHC 6.8.1 is available on request. Support for Glasgow parallel Haskell (→ 3.1.2) is currently being added to this version of the runtime environment. It is planned for the future to maintain a common parallel runtime environment for Eden, GpH, and other parallel Haskells.

Recent and Forthcoming Publications

- Jost Berthold: *Implicit and Explicit Parallel Functional Programming: Concepts and Implementation*, Dissertation (PhD thesis), Fachbereich Mathematik und Informatik, Philipps-Universität Marburg, April 2008.
- Alberto de la Encina: *Formalizando el proceso de depuración en programación funcional paralela y perezosa*, Tesis Doctoral (PhD thesis), Facultad de Ciencias Matemáticas, Universidad Complutense de Madrid, March 2008, in Spanish.
- Mischa Dieterle, Jost Berthold, and Rita Loogen: *Functional Implementation of a Distributed Work Pool Skeleton*, submitted.
- Oleg Lobachev and Rita Loogen: *Towards an Implementation of a Computer Algebra System in a Functional Language*, 9th International Conference on Artificial Intelligence and Symbolic Computation (AISC), Birmingham, July 2008, Springer LNAI, to appear.
- Abdallah Al Zain, Phil Trinder, Jost Berthold, Rita Loogen, Kevin Hammond, and Hans-Wolfgang Loidl: *Parallel Functional Middleware for Computational Algebra Systems*, Draft Proceedings of the Symposium on Trends In Functional Programming (TFP), Radboud Universiteit Nijmegen, May 2008.

- Mercedes Hidalgo-Herrero and Yolanda Ortega-Mallén: *Calculational Reasoning for Parallel Functional Programming*, Draft Proceedings of the Symposium on Trends In Functional Programming (TFP), Radboud Universiteit Nijmegen, May 2008.
- Oleg Lobachev, Jost Berthold, Mischa Dieterle, and Rita Loogen: *Parallel FFT Using Divide and Conquer Skeletons*, Draft Proceedings of the Symposium on Trends In Functional Programming (TFP), Radboud Universiteit Nijmegen, May 2008.
- Jost Berthold, Mischa Dieterle, Rita Loogen, and Steffen Priebe: *Hierarchical Master-Worker Skeletons*, Practical Aspects of Declarative Languages (PADL) 08, LNCS 4902, Springer 2008.
- Jost Berthold, Abyd Al-Zain, and Hans-Wolfgang Loidl: *Adaptive High-Level Scheduling in a Generic Parallel Runtime Environment*, Practical Aspects of Declarative Languages (PADL) 08, LNCS 4902, Springer 2008.
- Jost Berthold and Rita Loogen: *Visualising Parallel Functional Program Runs - Case Studies with the Eden Trace Viewer*, Parallel Computing: Architectures, Algorithms and Applications, Proceedings of the International Conference ParCo 2007, IOS Press 2007.

Further reading

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

3.1.4 XHaskell project

Report by:	Martin Sulzmann
Participants:	Kenny Zhuo Ming Lu

XHaskell is an extension of Haskell which combines parametric polymorphism, algebraic data types, and type classes with XDuce style regular expression types, subtyping, and regular expression pattern matching. The latest version can be downloaded via http://code.google.com/p/xhaskell/

Latest developments

We have fully implemented the system, which can be used in combination with the Glasgow Haskell Compiler. We have taken care to provide meaningful type error messages in case the static checking of programs fails. Our system also allows to defer some static checks until run-time.

We make use of GHC-as-a-library so that the XHaskell programmer can easily integrate her programs into existing applications and take advantage of the many libraries available in GHC. We also provide a convenient interface to the HaXML (→ 5.10.3) parser.

Kenny's thesis will be available shortly, describing in detail the formal underpinnings behind XHaskell.

3.1.5 HaskellActorJoin (previously: HaskellJoin)

Report by:	Martin Sulzmann

In this project, we extend Haskell with Erlang-style actors and Join-calculus style concurrency primitives. The HaskellJoin extension is described in an IFL'07 paper. See for details: http://taichi.ddns.comp.nus.edu.sg/taichiwiki/HaskellJoinRules.

The HaskellActor extension is described in a forthcoming COORDINATION'08 paper. The implementation can be downloaded via http://code.google.com/p/haskellactor/

Latest developments

We are currently working on revising the HaskellJoin implementation.

3.2 Related Languages

3.2.1 Curry

Report by:	Jan Christiansen
Participants:	Bernd Braßel, Michael Hanus, Wolfgang Lux, Sebastian Fischer, and others
Status:	active development

Curry is a functional logic programming language with Haskell syntax. In addition to the standard features of functional programming like higher-order functions and lazy evaluation, Curry supports features known from logic programming. This includes programming with non-determinism, free variables, constraints, declarative concurrency, and the search for solutions. Although Haskell and Curry share the same syntax, there is one main difference with respect to how function declarations are interpreted. In Haskell the order in which different rules are given in the source program has an effect on their meaning. In Curry, in contrast, the rules are interpreted as *equations*, and overlapping rules induce a non-deterministic choice and a search over the resulting alternatives. Furthermore, Curry allows to call functions with free variables as arguments so that they are bound to those values that are demanded for evaluation, thus providing for function inversion.

There are three major implementations of Curry. While the original implementation PAKCS (Portland Aachen Kiel Curry System) compiles to Prolog, MCC (Münster Curry Compiler) generates native code via a standard C compiler. The Kiel Curry System (KiCS) compiles Curry to Haskell aiming to provide nearly as good performance for the purely functional part as modern compilers for Haskell do. From these implementations only MCC will provide type classes in the near future. Type classes are not part of the current

definition of Curry, though there is no conceptual conflict with the logic extensions.

There have been research activities in the area of functional logic programming languages for more than a decade. Nevertheless, there are still a lot of interesting research topics regarding more efficient compilation techniques and even semantic questions in the area of language extensions like encapsulation and function patterns. Besides activities regarding the language itself, there is also an active development of tools concerning Curry (e.g., the documentation tool CurryDoc, the analysis environment CurryBrowser, the observation debuggers COOSy and iCODE, the debugger B.I.O. (http://www-ps.informatik.uni-kiel.de/currywiki/tools/oracle_debugger), EasyCheck (→ 4.3.3), and CyCoTest (→ 4.3.4)). Because Curry has a functional subset, these tools can canonically be transferred to the functional world.

Further reading

- http://www.informatik.uni-kiel.de/~curry
- http://www.informatik.uni-kiel.de/~pakcs
- http://danae.uni-muenster.de/~lux/curry
- http://www.informatik.uni-kiel.de/prog/mitarbeiter/bernd-brassel/projects
- http://www.informatik.uni-kiel.de/~curry/wiki

3.2.2 Agda

Report by:	Nils Anders Danielsson
Status:	Actively developed by a number of people

Do you crave for highly expressive types, but do not want to resort to type-class hackery? Then Agda might provide a view of what the future has in store for you.

Agda is a dependently typed functional programming language (developed using Haskell). The language has inductive families, i.e., GADTs which can be indexed by *values* and not just types. Other goodies include parameterised modules, mixfix operators, and an *interactive* Emacs 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 (effects, 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.

New since last time:

- A simple foreign function interface, which allows use of Haskell functions in Agda code.
- The libraries are steadily increasing in size.

Further reading

The Agda Wiki: http://www.cs.chalmers.se/~ulfn/Agda/

3.2.3 Epigram

Report by:	Conor McBride

Epigram is a prototype dependently typed functional programming language, equipped with an interactive editing and typechecking environment. High-level Epigram source code elaborates into a dependent type theory based on Zhaohui Luo's UTT. The definition of Epigram, together with its elaboration rules, may be found in "The view from the left" by Conor McBride and James McKinna (JFP 14 (1)).

A new version, Epigram 2, based on Observational Type Theory (see "Observational Equality, Now!" by Thorsten Altenkirch, Conor McBride, and Wouter Swierstra) is in preparation.

Motivation

Simply typed languages have the property that any subexpression of a well typed program may be replaced by another of the same type. Such type systems may guarantee that your program will not crash your computer, but the simple fact that True and False are always interchangeable inhibits the expression of stronger guarantees. Epigram is an experiment in freedom from this compulsory ignorance.

Specifically, Epigram is designed to support programming with inductive datatype families indexed by data. Examples include matrices indexed by their dimensions, expressions indexed by their types, search trees indexed by their bounds. In many ways, these datatype families are the progenitors of Haskell's GADTs, but indexing by data provides both a conceptual simplification — the dimensions of a matrix are *numbers* — and a new way to allow data to stand as *evidence* for the properties of other data. It is no good representing sorted lists if comparison does not produce evidence of ordering. It is no good writing a type-safe interpreter if one's typechecking algorithm cannot produce well-typed terms.

Programming with evidence lies at the heart of Epigram's design. Epigram generalises constructor pattern matching by allowing types resembling induction principles to express as how the inspection of data may affect both the flow of control at run time and the text and type of the program in the editor. Epigram extracts patterns from induction principles and induction principles from inductive datatype families.

History

James McKinna and Conor McBride designed Epigram in 2001, whilst based at Durham, working with Zhao-

hui Luo and Paul Callaghan. McBride's prototype implementation of the language, "Epigram 1" emerged in 2004: it is implemented in Haskell, interfacing with the xemacs editor. This implementation effort involved inventing a number of new programming techniques which have found their way into the Haskell community at large: central components of Control.Applicative and Data.Traversable started life in the source code for Epigram.

Following the Durham diaspora, James McKinna and Edwin Brady went to St. Andrews, where they continued their work on phase analysis and efficient compilation of dependently typed programs. More recently, with Kevin Hammond, they have been studying applications of dependent types to resource-aware computation in general, and network protocols in particular.

Meanwhile, Conor McBride went to Nottingham to work with Thorsten Altenkirch. They set about redesigning Epigram's underlying type theory, radically changing its treatment of logical propositions in general, and *equality* in particular, making significant progress on problems which have beset dependent type theories for decades.

The Nottingham duo grew into a strong team of enthusiastic researchers. Peter Morris successfully completed a PhD on generic programming in Epigram and is now a research assistant: his work has led to the redesign of Epigram's datatype language. Nicolas Oury joined from Paris as a postdoctoral research fellow, and is now deeply involved in all aspects of design and implementation. PhD students James Chapman and Wouter Swierstra are working on Epigram-related topics, studying formalised metatheory and effectful programming, respectively. Meanwhile, Nottingham research on *containers*, involving Neil Ghani, Peter Hancock, and Rawle Prince, together with the Epigram team, continues to inform design choices as the language evolves.

Epigram 1 was used successfully by Thorsten Altenkirch, Conor McBride, and Peter Hancock in an undergraduate course on Computer Aided Formal Reasoning http://www.e-pig.org/darcs/g5bcfr/. It has also been used in a number of graduate-level courses.

James McKinna is now at Radboud University, Nijmegen; Edwin Brady is still at St. Andrews; Thorsten Altenkirch, Peter Morris, Nicolas Oury, James Chapman, and Wouter Swierstra are still in Nottingham; Conor McBride has left academia. All are still contributing to the Epigram project.

Current Status

Epigram 2 is based on a radical redesign of our underlying type theory. The main novelties are

- a *bidirectional* approach to typechecking, separating syntactically the terms whose types are inferred from those for which types are pushed in — with stronger guarantees of prior type information, we can reduce clutter in terms and support greater overloading;

- explicit separation of *propositions* and *sets*, ensuring that proofs never influence control-flow and can be erased at run-time;

- a *type-directed* approach to propositional equality, comparing functions extensionally, records componentwise, data by construction, and proofs trivially — we shall soon support equality for codata by bisimulation and for quotients by whatever you want;

- three *closed universes* of data structures, finite enumerations, record types, and inductive datatypes, each with its datatype of type descriptions — this supports generic programming over all of Epigram 2's data structures and removes the need for any means of "making new stuff" other than definition.

Nicolas Oury, Peter Morris, and Conor McBride have implemented this theory, together with a system supporting interactive construction (and destruction) within it. This the engine which will drive Epigram 2: we plan to equip it with human-accessible controls and release it for the benefit of the curious, shortly. With this in place, we shall reconstruct the Epigram source language and its elaboration mechanism: constructs in source become constructions in the core.

There is still a great deal of work to do. We need to incorporate the work from Edwin Brady and James McKinna on type erasure and efficient compilation; we need to bring out and exploit the container structure of data; we need to support programming with effects (including non-termination); we need a declarative proof language, as well as a functional programming language.

The Epigram project relies on Haskell, its libraries, and tools such as Alex (→ 4.1.1), Happy (→ 4.1.2), bnfc, Cabal (→ 5.1), and Darcs (→ 6.1.1). We have recently developed tools for assembling the modules corresponding to each component of the Epigram system from files corresponding to each feature of the Epigram language: this may prove useful to others, so we hope to clean them up and release them. Meanwhile, as Haskell itself edges ever closer to dependent types, the Epigram project has ever more to contribute, in exploration of the design space, in the development of implementation technique, and in experimentation with the pragmatics of programming with such power and precision.

Epigram source code and related research papers can be found on the web at http://www.e-pig.org and its community of experimental users communicates via the mailing list ⟨epigram@durham.ac.uk⟩. The current, rapidly evolving state of Epigram 2 can be found at http://www.e-pig.org/epilogue/.

3.3 Type System / Program Analysis

3.3.1 Uniqueness Typing

Report by:	Edsko de Vries
Participants:	Rinus Plasmeijer, David M Abrahamson
Status:	ongoing

An important feature of pure functional programming languages is referential transparency. A consequence of referential transparency is that functions cannot be allowed to modify their arguments, *unless* it can be guaranteed that they have the sole reference to that argument. This is the basis of uniqueness typing.

We have been developing a uniqueness type system based on that of the language *Clean* but with various improvements: no subtyping is required, the type language does not include inequality constraints (types in Clean often involve implications between uniqueness attributes), and types and uniqueness attributes are both considered types (albeit of different kinds). This makes the type system sufficiently similar to standard Hindley/Milner type systems that (1) standard inference algorithms can be applied, and (2) modern extensions such as arbitrary rank types and generalised algebraic data types (GADTs) can easily be incorporated.

Although our type system is developed in the context of the language Clean, it is also relevant to Haskell because the core uniqueness type system we propose is very similar to Haskell's core type system.

Further reading

- Edsko de Vries, Rinus Plasmeijer, and David Abrahamson, "Uniqueness Typing Simplified", in *Proceedings of IFL 2007* (to appear in the LNCS series).
- Edsko de Vries, Rinus Plasmeijer, and David Abrahamson, "Uniqueness Typing Redefined", in *Z. Horváth, V. Zsók, and Andrew Butterfield (Eds.): IFL 2006, LNCS 4449.*

3.3.2 Free Theorems for Haskell

Report by:	Janis Voigtländer
Participants:	Sascha Böhme, Florian Stenger

Free theorems are statements about program behaviour 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. Recent papers concerning program transformations are "Semantics and Pragmatics of New Shortcut Fusion Rules" (FLOPS'08) and "Asymptotic Improvement of Computations over Free Monads" (MPC'08).

Also on the practical side, we maintain a library and tools for generating free theorems from Haskell types, originally implemented by Sascha Böhme. Both the library and a shell-based tool are now available from Hackage (as free-theorems-0.2 and ftshell-0.2, respectively). There is also a web-based tool at http://linux.tcs.inf.tu-dresden.de/~voigt/ft. General features include:

- three different language subsets to choose from
- equational as well as inequational free theorems
- relational free theorems as well as specialisations down to function level
- support for algebraic data types, type synonyms and renamings, type classes

While the web-based tool is restricted to algebraic data types, type synonyms, and type classes from Haskell standard libraries, the shell-based tool also enables the user to declare their own algebraic data types and so on, and then to derive free theorems from types involving those. A distinct new feature of the web-based tool is to export the generated theorems in PDF format.

Further reading

http://wwwtcs.inf.tu-dresden.de/~voigt/project/

4 Tools

4.1 Scanning, Parsing, Transformations

4.1.1 Alex version 2

Report by:	Simon Marlow
Status:	stable, maintained

Alex is a lexical analyser generator for Haskell, similar to the tool lex for C. Alex takes a specification of a lexical syntax written in terms of regular expressions, and emits code in Haskell to parse that syntax. A lexical analyser generator is often used in conjunction with a parser generator, such as Happy (→ 4.1.2), to build a complete parser.

The latest release is version 2.2, released November 2007. Alex is in maintenance mode, we do not anticipate any major changes in the near future.

Changes in version 2.2:

- ByteString wrappers: use Alex to lex ByteStrings (→ 5.5.1) directly.

Further reading

http://www.haskell.org/alex/

4.1.2 Happy

Report by:	Simon Marlow
Status:	stable, maintained

Happy is a tool for generating Haskell parser code from a BNF specification, similar to the tool Yacc for C. Happy also includes the ability to generate a GLR parser (arbitrary LR for ambiguous grammars).

The latest release is 1.17, released 22 October 2007.

Changes in version 1.17:

- Works with GHC 6.8.x, and requires Cabal 1.2.
- Fix serious mistake in error handling (the "parE" bug)
- Some performance improvements to Happy itself

Further reading

Happy's web page is at http://www.haskell.org/happy/. Further information on the GLR extension can be found at http://www.dur.ac.uk/p.c.callaghan/happy-glr/.

4.1.3 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 which makes it easy to write *catamorphisms* (that is, functions that do to any datatype what *foldr* does to lists). You can define tree walks using the intuitive concepts of *inherited* and *synthesised attributes*, while keeping the full expressive power of Haskell. The generated tree walks are *efficient* in both space and time.

New features are support for polymorphic abstract syntax and higher-order attributes. With polymorphic abstract syntax, the type of certain terminals can be parameterised. Higher-order attributes are useful to incorporate computed values as subtrees in the AST.

The system is in use by a variety of large and small projects, such as the Haskell compiler EHC, the editor Proxima for structured documents, the Helium compiler (→ 2.4), the Generic Haskell compiler, and UUAG itself. The current version is 0.9.6 (April 2008), is extensively tested, and is available on Hackage.

We are currently improving the documentation, and plan to introduce an alternative syntax that is closer to the Haskell syntax.

Further reading

- http://www.cs.uu.nl/wiki/bin/view/HUT/AttributeGrammarSystem
- http://hackage.haskell.org/cgi-bin/hackage-scripts/package/uuagc-0.9.6

4.2 Documentation

4.2.1 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 the 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 (→ 5.1), 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).

The latest release is version 2.1.0, released May 1 2008.

Changes since the 0.9 release:

- As a result of a Google Summer of Code project, Haddock now uses the GHC API (→ 2.1) as its front end. This means that Haddock can process any GHC-compatible Haskell code.

Changes since the 0.8 release:

- Thanks to Neil Mitchell, the index page generated by Haddock now has a search box, and the list is dynamically updated as you type.

Future plans

Currently, Haddock ignores comments on some language constructs like GADTs and Associated Type synonyms. Of course, the plan is to support comments for these constructs in the future. Haddock is also slightly more picky on where to put comments compared to the 0.x series. We want to fix this as well. Both of these plans require changes to the GHC parser. We want to investigate to what degree it is possible to decouple comment parsing from GHC and move it into Haddock, to not be bound by GHC releases.

Further reading

- There is a TODO list of outstanding bugs and missing features, which can be found here: http://code.haskell.org/haddock/TODO
- Haddock's homepage is here: http://www.haskell.org/haddock/

4.2.2 lhs2TeX

Report by:	Andres Löh
Status:	stable, maintained

This tool by Ralf Hinze and Andres Löh is a preprocessor that transforms literate Haskell code into LaTeX documents. The output is highly customisable 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, and does not restrict the user to Haskell 98.

The program is stable and can take on large documents.

Since the last report, version 1.13 has been released. It is compatible with GHC 6.8 and Cabal 1.2, but not yet with development versions of Cabal. Maintenance will continue in the future, releases will appear as needed. No major development is planned at the moment, although I have some vague ideas for substantial improvement.

Further reading

- http://www.cs.uu.nl/~andres/lhs2tex
- https://svn.cs.uu.nl:12443/viewcvs/lhs2TeX/lhs2TeX/trunk/

4.3 Testing and Debugging

4.3.1 SmallCheck

Report by:	Colin Runciman
Status:	Version 0.3 May 2008

SmallCheck is a one-module lightweight testing library. It adapts QuickCheck's ideas of type-based generators for test data and a class of testable properties. But instead of testing a sample of randomly generated values, it tests properties for all the finitely many values up to some depth, progressively increasing the depth used. Among other advantages, generators for user-defined types can follow a simple pattern and are automatically derivable.

The two significant developments in Version 0.3 are (1) new variants of existential quantifiers requiring uniqueness — two witnesses are reported when uniqueness fails, and (2) a new method for generating functions with functional arguments — avoiding previous over-generation in some cases. There are other more minor improvements.

SmallCheck is freely available for downloading from http://www.cs.york.ac.uk/fp/smallcheck0.3.tar.

For the first-order universally quantified subset of SmallCheck properties, the pruning principles implemented in Lazy SmallCheck (→ 4.3.2) often allow deeper testing at the same computational cost. A likely next development is a fuller combination of techniques from these two testing libraries.

4.3.2 Lazy SmallCheck

Report by:	Matthew Naylor
Participants:	Fredrik Lindblad, Colin Runciman
Status:	experimental

Lazy SmallCheck is a library for testing program properties. Unlike QuickCheck and SmallCheck (→ 4.3.1), it generates *partially-defined* inputs that are progressively refined as demanded by the property under test. The key observation is that if a property evaluates to True or False for a partially-defined input then it would also do so for all refinements of that input. By not generating such refinements, Lazy SmallCheck may test

the same input-space as SmallCheck using significantly fewer tests.

A talk about Lazy SmallCheck was given at Fun in the Afternoon York, and the slides are available, along with an initial implementation, at the URL below. Since the last HCAR, we have made a simpler implementation, now supporting Fredrik's parallel conjunction operator. The problem with a sequential conjunction is that, when applied to a partially-defined input, it will crash (i.e., demand more input than is given) if the first conjunct crashes, even if the second conjunct is falsified. A parallel conjunction, in contrast, is falsified if *any* conjunct is falsified, even if the other conjucts crash. This increases pruning opportunities, and "when used, one is not obliged to tweak the order of conjuncts in the property" (Fredrik Lindblad, TFP 2007).

Support for existential quantifiers and function generation a la SmallCheck is under consideration. A new release will hopefully be made sometime during the year.

Further reading

http://www.cs.york.ac.uk/~mfn/lazysmallcheck/

4.3.3 EasyCheck

Report by:	Jan Christiansen
Participants:	Sebastian Fischer
Status:	experimental

EasyCheck is an automatic test tool like QuickCheck or SmallCheck (→ 4.3.1). It is implemented in the functional logic programming language Curry (→ 3.2.1). Although simple test cases can be generated from nothing but type information in all mentioned test tools, users have the possibility to define custom test-case generators — and make frequent use of this possibility. Nondeterminism — the main extension of functional-logic programming over Haskell — is an elegant concept to describe such generators. Therefore it is easier to define custom test-case generators in EasyCheck than in other test tools. If no custom generator is provided, test cases are generated by a free variable which non-deterministically yields all values of a type. Moreover, in EasyCheck, the enumeration strategy is independent of the definition of test-case generators. Unlike QuickCheck's strategy, it is complete, i.e., every specified value is eventually enumerated if enough test cases are processed, and no value is enumerated twice. SmallCheck also uses a complete strategy (breadth-first search) which EasyCheck improves w.r.t. the size of the generated test data. EasyCheck is distributed with the Kiel Curry System (KiCS).

Further reading

http://www-ps.informatik.uni-kiel.de/currywiki/tools/easycheck

4.3.4 CyCoTest

Report by:	Sebastian Fischer
Participants:	Herbert Kuchen
Status:	experimental

The Curry Coverage Tester CyCoTest (pronounced like psycho test) aims at testing declarative programs to the bone. Unlike black-box test tools like QuickCheck, it does not generate test cases from type information or additional specifications. It rather uses the demand of the program under test to *narrow* test cases lazily. Narrowing is a generalisation of reduction that allows to compute with partial information. Evaluating a program with narrowing and initially uninstantiated input binds the input as much as demanded by the computation and non-deterministically computes a corresponding result for each binding. The generated pairs of in- and output form a set of test cases that reflects the demand of the tested program.

The generated set of test cases can either be checked by hand or using properties, i.e., functions with a Boolean result. Using properties is convenient, but sometimes it is hard to come up with a complete formal specification of the tested program. Hence, errors might remain undetected if an incomplete property is used to evaluate the test cases. In order to lower the burden of manual checking, we employ control- and data-flow coverage information to minimise the set of generated test cases. Test cases that do not cause new code coverage are considered redundant and need not be shown to the user. Although this bears the risk of eliminating test cases that expose a bug, experiments indicate that the employed coverage criteria suffice to expose bugs in practice.

CyCoTest is implemented in and for the functional logic programming language Curry (→ 3.2.1), which provides narrowing for free. A Haskell implementation would be possible using ideas from the Kiel Curry System (KiCS), which translates Curry programs into Haskell programs.

Further reading

http://www-ps.informatik.uni-kiel.de/currywiki/tools/cycotest

4.3.5 Hat

Report by:	Olaf Chitil
Participants:	Malcolm Wallace
Status:	maintenance

The Haskell tracing system Hat is based on the idea that a specially compiled Haskell program generates a trace file alongside its computation. This trace can be viewed in various ways with several tools. Some views are similar to classical debuggers for imperative languages, some are specific to lazy functional language features or particular types of bugs. All tools interoperate and use a similar command syntax.

Hat can be used both with nhc98 (→ 2.2) and GHC (→ 2.1). Hat was built for tracing Haskell 98 programs, but it also supports some language extensions (FFI, MPTC, fundeps, hierarchical libs). A tutorial explains how to generate traces, how to explore them, and how they help to debug Haskell programs.

During the last half year only small bug fixes were committed to the Darcs repository, but several other updates are also planned for the near future, including new and improved trace-browsers. A recent student project completed a Java-GUI viewer for traces, based on the idea of timelines and search. We hope this can be added to the repository soon.

Further reading

- http://www.haskell.org/hat
- `darcs get` http://darcs.haskell.org/hat
- Tracing and Debugging Functional Programs: http://www.cs.kent.ac.uk/~oc/tracing.html

4.4 Development

4.4.1 Hoogle — Haskell API Search

Report by:	Neil Mitchell
Status:	v3.0

Hoogle is an online Haskell API search engine. It searches the functions in the various libraries, both by name and by type signature. When searching by name, the search just finds functions which contain that name as a substring. However, when searching by types it attempts to find any functions that might be appropriate, including argument reordering and missing arguments. The tool is written in Haskell, and the source code is available online. Hoogle is available as a web interface, a command line tool, and a lambdabot (→ 6.8.1) plugin.

The development of Hoogle has been slow lately, due to the author writing a PhD thesis. However, this summer Hoogle will become a full-time project with funding from the Google Summer of Code. Expect to see Hoogle v4.0 before the summer is over.

Further reading

http://haskell.org/hoogle

4.4.2 Leksah, Haskell IDE

Report by:	Jürgen Nicklisch-Franken
Status:	in development

Leksah is a Haskell IDE written in Haskell based on Gtk+ and gtk2hs (→ 5.8.1). Leksah is a practical tool to support the Haskell development process. It is platform independent and should run on any platform where GTK+, gtk2hs, and GHC can be installed. (It is currently being tested on Windows and Linux but it should work on the Mac. It only works with GHC.)

There are compelling reasons for a Haskell IDE written in Haskell. First and most importantly, Haskell is different from mainstream imperative and object oriented languages and a dedicated IDE may exploit this specialness. Second the integration with an existing tool written in a different language has to solve the problem of integration of different programming languages/paradigms.

Currently Leksah offers features like jumping to definition for a name, integration of Cabal (→ 5.1) for building, Haskell source editor with "source candy", configurable keymaps, ... This list will (hopefully) expand quickly.

The development of Leksah started in June 2007 and the first alpha version was released February 2008. Contributions of all kind are welcome.

Further reading

http://leksah.org/

4.4.3 EclipseFP — Haskell support for the Eclipse IDE

Report by:	Leif Frenzel
Status:	alpha

The Eclipse platform is an extremely extensible framework for IDEs, developed by an Open Source Project. Our project extends it with tools to support Haskell development.

The aim is to develop an IDE for Haskell that provides the set of features and the user experience known from the Eclipse Java IDE (the flagship of the Eclipse project), and integrates a broad range of Haskell development tools. Long-term goals include support for language-aware IDE features, like refactoring and structural search.

Over the past year, a new subproject called Cohatoe has developed a framework that allows us to implement Eclipse Plugins partly in Haskell. We are currently reimplementing and extending EclipseFP functionality in Haskell, using libraries such as Cabal (→ 5.1) and the GHC API (→ 2.1). The goal is to release a new version, EclipseFP 2, this summer.

Further reading

- http://eclipsefp.sf.net
- http://leiffrenzel.de/eclipse/wiki/
- http://lists.sourceforge.net/lists/listinfo/eclipsefp-develop

4.4.4 yi

Report by:	Jean-Philippe Bernardy
Participants:	Don Stewart
Status:	active development

Yi is a project to write a Haskell-extensible editor. Yi is structured around a purely functional editor core, such that most components of the editor can be overridden by the user, using configuration files written in Haskell.

Yi has been converted to the Cabal build system, which makes it easier to build and experiment with.

Yi features:

- Keybindings for emacs and vim, written as extensible parsers;
- Vty and Gtk2Hs frontends;
- Syntax highlighting for Haskell and other languages;
- XMonad-style static configuration;
- Support of Linux, MacOS, and Windows platforms.

We are currently working on the following fronts:

- Integration with Cabal and GHC API;
- Syntax-aware support of Haskell;
- Pango, cocoa frontends

Further reading

- Documentation can be found at: http://haskell.org/haskellwiki/Yi
- The source repository is available: `darcs get` http://code.haskell.org/yi/

4.4.5 HaRe — The Haskell Refactorer

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

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

Our project, *Refactoring Functional Programs*, has as its major goal to build a tool to support refactorings in Haskell. The HaRe tool is now in its fourth major release. HaRe supports full Haskell 98, and is integrated with Emacs (and XEmacs) and Vim. All the refactorings that HaRe supports, including renaming, scope change, generalisation, and a number of others, are *module aware*, so that a change will be reflected in all the modules in a project, rather than just in the module where the change is initiated. The system also contains a set of data-oriented refactorings which together transform a concrete `data` type and associated uses of pattern matching into an abstract type and calls to assorted functions. The latest snapshots support the hierarchical modules extension, but only small parts of the hierarchical libraries, unfortunately. An informal release of HaRe 0.4 that was recently released works with GHC 6.6.1 and GHC 6.8.2, but not GHC 6.4; earlier releases work with 6.4.*.

In order to allow users to extend HaRe themselves, HaRe includes an API for users to define their own program transformations, together with Haddock (→ 4.2.1) documentation. Please let us know if you are using the API.

There have been some recent developments for adding program slicing techniques to HaRe. These techniques include a refactoring to split functions returning tuples into separate definitions, and to also put them back together again. A number of new data-type based and structural refactorings have been added to HaRe. Some of these refactorings make use of the GHC type checker to make the refactorings type-aware. These new refactorings include: adding and removing a constructor; adding and removing a field; and introduction of pattern matches and case analysis. Structural refactorings include: Conversion between let and where, and folding and unfolding of as-patterns.

A snapshot of HaRe is available from our webpage, as are recent presentations from the group (including LDTA 05, TFP05, SCAM06), and an overview of recent work from staff, students, and interns. Among this is an evaluation of what is required to port the HaRe system to the GHC API (→ 2.1), and a comparative study of refactoring Haskell and Erlang programs.

The final report for the project appears there, too, together with an updated refactoring catalogue and the latest snapshot of the system. Huiqing's PhD thesis on refactoring Haskell programs is now available online from our project webpage.

Further reading

http://www.cs.kent.ac.uk/projects/refactor-fp/

4.4.6 Haskell Mode Plugins for Vim

Report by:	Claus Reinke
Participants:	Haskell & Vim users
Status:	maintenance mode

My Haskell mode plugins for Vim seem to have become quite popular. They collect several scripts that offer functionality based on GHCi, on Haddock-generated documentation (→ 4.2.1), and on Vim's own configurable program editing support. This includes several

insert mode completions (based on identifiers available via currently imported modules, on identifiers appearing in the central Haddock indices, on tag files, or on words appearing in current and imported sources), quickfix mode (call compiler, list errors, jump to error locations), inferred type tooltips, various editing helpers (insert import statement, type declaration or module qualifier for id under cursor, expand implicit into explicit import statement, add option and language pragmas, ...), and direct access to the Haddocks for the id under cursor.

A very incomplete screenshot tour of Vim's IDE functions, as instantiated for Haskell, provides an overview of what is available (for more general information, see Vim's excellent built-in `:help`, or browse the help files online at http://vimdoc.sourceforge.net/htmldoc/usr_toc.html; for more and current details of Haskell mode features, see the `haskellmode.txt` help file at the project site).

Both alternative and complementary Haskell-related plugins for Vim exist — please add links to your own tricks and tips at haskell.org (syntax-colouring works out of the box, other scripts deal with indentation, ...). I hope these plugins might be useful to some of you (please let me know if anything does not work as advertised!), and might even motivate some of you to give Vim a try. It is really not as if Vim (or Emacs, for that matter) did not have more IDE functionality than most of us ever use, it is more that there is so much of it to learn and to fine-tune to your personal preferences.

The haskellmode plugins for Vim are currently in maintenance mode, with infrequent updates and bug fixes, and the occasional new feature (CamelCase-based abbreviations for insert-mode completion the most recent example).

Further reading

- Haskell Mode Plugins for Vim: http://www.cs.kent.ac.uk/~cr3/toolbox/haskell/Vim/
- A short tour of some Vim support for Haskell editing (screenshots): http://www.cs.kent.ac.uk/~cr3/toolbox/haskell/Vim/vim.html
- haskell.org section listing these and other Vim files: http://www.haskell.org/haskellwiki/Libraries_and_tools/Program_development#Vim

4.4.7 :def and .ghci (previously: dot.ghci)

Report by:	Claus Reinke
Status:	old, but underappreciated

No matter how fast GHCi keeps improving, users still keep suggesting new commands for it, and with the increasing amounts of information available in GHCi, it can sometimes be difficult to find the interesting bits, either in the documentation, or indeed in command outputs. The usual approach is to add feature requests to the ticket tracker and hope that someone will get round to implementing them, or to get the GHC sources and start contributing code.

Quite often, however, users ask for GHCi features that they could (relatively) easily define themselves, using some of the less well known features of GHCi.

In an email to the haskell-cafe last September, I gave one demonstration of this approach in the form of a mini-tutorial, starting with simple things like platform-independent `:pwd`/`:ls`, then laying the groundwork for more complex commands by defining `:redir <var> <cmd>`, a command that redirects the output of `<cmd>`, binding it to variable `<var>`. Based on `:redir`, we can then define `:grep <pat> <cmd>`, to filter the output of `<cmd>` for a pattern `<pat>` (think of finding the help entries related to breakpoints, or the variants of fold appearing in `:browse Prelude`). Taking some examples from the GHCi ticket tracker and from Hugs' commands, there is also `:find <id>` (open the source for the definition of `<id>`), `:b` (`:browse` first module listed in `:show modules`), and `:le <mod>` (load module `<mod>`, edit location of first error, if any).

The commands are self-documenting, can be listed and removed as a group, and should give a good starting point for your own experiments with GHCi's `:def`. Since the email, which targeted GHC 6.6 and later, a version for GHC 6.4.1 was added for those who needed to work with an old installation (the commands in that version differ slightly, to account for 6.4.1's limitations). Please let me know if you find this useful, and remember to share your own GHCi tricks and tips!

You can find (and contribute!) other suggestions for `.ghci` files on this Haskell wiki page: http://haskell.org/haskellwiki/GHC/GHCi.

Further reading

- http://www.haskell.org/pipermail/haskell-cafe/2007-September/032260.html
- http://www.cs.kent.ac.uk/~cr3/toolbox/haskell/#dot.ghci

4.4.8 DarcsWatch

Report by:	Joachim Breitner
Status:	working, in early development

DarcsWatch is a tool to track the state of Darcs (→ 6.1.1) 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. Some notable features are:

- Reads both darcs1 and darcs2.0 (hashed) format repositories.

- Displays patch summaries per user and per repository.
- Patch diff can be reviewed directly.
- Download link for each patch, to apply without searching for the mail.
- Knows about inverse and amend-recorded patches and uses them to consider patches obsolete.
- Patches can by marked obsolete or rejected by email commands.
- Can be subscribed to a project mailing list, or be used as a CC recipient for darcs bundles.

Further reading

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

4.4.9 cpphs

Report by:	Malcolm Wallace
Status:	stable, maintained

Cpphs is a robust drop-in Haskell replacement for the C pre-processor. It has a couple of benefits over the traditional cpp — you can run it in Hugs when no C compiler is available (e.g., on Windows); and it understands the lexical syntax of Haskell, so you do not get tripped up by C-comments, line-continuation characters, primed identifiers, and so on. (There is also a pure text mode which assumes neither Haskell nor C syntax, for even greater flexibility.)

Cpphs can also unliterate `.lhs` files during preprocessing, and you can install it as a library to call from your own code, in addition to the stand-alone utility.

Current release is 1.4: there have been no changes in the last six months, indicating (one hopes) that it is now relatively bug-free.

Further reading

http://haskell.org/cpphs

5 Libraries

5.1 Cabal and Hackage

Report by: Duncan Coutts

Background

The Haskell Cabal is a Common Architecture for Building Applications and Libraries. It is an API distributed with GHC (→ 2.1), nhc98 (→ 2.2), and Hugs which allows a developer to easily build and distribute packages.

Hackage (Haskell Package Database) is an online database of packages which can be interactively queried via the website and client-side software such as cabal-install. From Hackage, an end-user can download and install Cabal packages.

Recent progress

We are preparing for a release of Cabal-1.4 and the cabal-install tool. It is expected that all packages that worked with Cabal-1.2 will work with Cabal-1.4. The Cabal-1.4 release will include a large number of incremental improvements and bug fixes. The major feature for the cabal-install release is that it can serve as the primary command line front end to the Cabal/Hackage system. There will be no need to use the `runhaskell Setup.hs` interface any more.

The last year has seen Hackage take off. It has grown from a handful of packages to nearly 600 (with over 1200 releases). The Hackage website has seen a number of improvements including reporting which versions of ghc each package builds with, who uploaded each package, and proper support for Unicode in package meta-data.

Hackage also now applies somewhat stricter rules about package uploads. Uploading the same version of a package more than once is no longer allowed because we expect package content to be stable. It also checks for some common problems in package descriptions and will warn or reject as appropriate. You can run the same checks locally using `cabal check`.

Google Summer of Code projects

We are very lucky to have got two Google Summer of Code projects related to Cabal and Hackage. Andrea Vezzosi is doing a project to build a "make-like" dependency framework for the Cabal library. This will enable Cabal to do proper dependency based rebuilds and support pre-processors like c2hs correctly. It also leads the way towards not depending on `ghc -make` for building Haskell code and to do parallel builds.

Neil Mitchell is working on Hoogle 4 (→ 4.4.1), which we aim to use as the primary search interface on the Hackage website. The aim is to be able to search the content of every package; not just the API but also the documentation and package meta-data.

Looking forward

There is huge potential for Hackage to help us manage and improve the community's package collection. We are nearing the point where cabal-install will be able to report build results to the Hackage server. This should provide us with a huge amount of data on which packages work in which environments and configurations. More generally there is the opportunity to collect all sorts of useful information on the quality of packages. Hopefully we can evolve Hackage and associated clients into the kind of infrastructure we need to assemble collections of packages into high quality "batteries included"-style Haskell platform releases.

To help us in the next round of development work it would be enormously helpful to know from our users what their most pressing problems are with Cabal and Hackage. You probably have a favourite Cabal bug or limitation. Take a look at our bug tracker. Make sure the problem is reported there and properly described. Comment on the ticket to tell us how much of a problem the bug is for you. Add yourself to the ticket's *cc* list so we can discuss requirements and keep you informed on progress. For feature requests it is very helpful if there is a description of how you would expect to interact with the new feature.

People

We would like to thank the large number of people who contributed to the last round of development work. I hope that attests to the fact that it is not too hard to get involved. Thanks also to the people who have followed development and reported bugs and feature requests.

Further reading

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

5.2 Auxiliary Libraries

5.2.1 libmpd

Report by:	Ben Sinclair
Participants:	Joachim Fasting
Status:	active

LIBMPD is a binding to the MPD music playing daemon's network protocol. While its interface has mostly stabilised and is ready to use, there is still some experimentation going on and we are seeking feedback on the API's design.

The latest release is 0.3.0, which has support for UTF-8 encoded data, a number of QuickCheck and unit tests, and fixes some oddities in the interface. We plan to look into speeding up the network IO soon.

Further reading

The development web page is at http://turing.une.edu.au/~bsinclai/code/libmpd-haskell/ and MPD can be found at http://www.musicpd.org/.

5.2.2 gravatar

Report by:	Don Stewart
Status:	active development

Gravatars (http://gravatar.com) are globally unique images associated with an email address, widely used in social networking sites. This library lets you find the URL of a gravatar image associated with an email address.

Further reading

- Source and documentation can be found on Hackage.
- The source repository is available:
  `darcs get` http://code.haskell.org/~dons/code/gravatar/

5.2.3 mersenne-random

Report by:	Don Stewart
Status:	active development

The Mersenne twister is a pseudorandom number generator developed by Makoto Matsumoto and Takuji Nishimura that is based on a matrix linear recurrence over a finite binary field. It provides for fast generation of very high quality pseudorandom numbers.

This library uses SFMT, the SIMD-oriented Fast Mersenne Twister, a variant of Mersenne Twister that is much faster than the original. It is designed to be fast when it runs on 128-bit SIMD. It can be compiled with either SSE2 OR PowerPC AltiVec support, to take advantage of these instructions.

By default the period of the function is $2^{19937} - 1$, however, you can compile in other defaults. Note that this algorithm on its own is not cryptographically secure.

Further reading

- Source and documentation can be found on Hackage.
- The source repository is available:
  `darcs get` http://code.haskell.org/~dons/code/mersenne-random/

5.2.4 cmath

Report by:	Don Stewart
Status:	active development

cmath is a complete, efficient binding to the standard C math.h library, for Haskell.

Further reading

- Source and documentation can be found on Hackage.
- The source repository is available:
  `darcs get` http://code.haskell.org/~dons/code/cmath/

5.2.5 hmatrix (previously: GSLHaskell)

Report by:	Alberto Ruiz
Status:	stable, maintained

hmatrix is a simple library for linear algebra and numerical computations, internally implemented using GSL, BLAS, and LAPACK. It is available from Hackage.

Most linear algebra functions mentioned in GNU-Octave's Quick Reference are available both for real and complex matrices: eig, svd, chol, qr, hess, schur, inv, pinv, expm, norm, and det. There are also functions for numeric integration and differentiation, nonlinear minimisation, polynomial root finding, and more than 200 GSL special functions. A brief manual is available at the URL below.

Recent developments include support for 64bit machines, improved testing, and support for Intel's MKL implementation of BLAS and LAPACK.

Further reading

http://alberrto.googlepages.com/gslhaskell

5.2.6 HPDF

Report by:	alpheccar
Status:	Continuous development

HPDF is an Haskell library allowing to generate PDF documents. HPDF is supporting several features of the PDF standard like outlines, multi-pages, annotations, actions, image embedding, shapes, patterns, text.

In addition to the standard PDF features, HPDF is providing some typesetting features built on top of the PDF core. With HPDF, it is possible to define complex styles for sentences and paragraphs. HPDF is implementing an optimum-fit line breaking algorithm a bit like the TeX one and HPDF is using the standard Liang hyphenation algorithm.

HPDF is at version 1.3. It is progressing continuously. HPDF is available on Hackage.

There are several missing features: the only supported fonts are the standard PDF ones. A next version should support TrueType and different character encodings. For support of Asian languages, I will ask for help in the Haskell community.

I also plan to define an API easing the definition of complex layouts (slides, books). Currently the layout has to be coded by hand, but it is already possible to build complex things.

The documentation is a bit weak and will have to be improved.

Further reading

http://www.alpheccar.org

5.2.7 The Neon Library

Report by:	Jurriaan Hage

As part of his master thesis work, Peter van Keeken implemented a library to data mine logged Helium (→ 2.4) programs to investigate aspects of how students program Haskell, how they learn to program, and how good Helium is in generating understandable feedback and hints. The software can be downloaded from http://www.cs.uu.nl/wiki/bin/view/Hage/Neon, which also gives some examples of output generated by the system. The downloads only contain a small sample of loggings, but it will allow programmers to play with it.

5.2.8 uniplate

Report by:	Neil Mitchell

Uniplate is a boilerplate removal library, with similar goals to the original Scrap Your Boilerplate work. It requires fewer language extensions, and allows more succinct traversals with higher performance than SYB. A paper including many examples was presented at the Haskell Workshop 2007.

If you are writing a compiler, or any program that operates over values with many constructors and nested types, you *should* be using a boilerplate removal library. This library provides a gentle introduction to the field, and can be used practically to achieve substantial savings in code size and maintainability.

Further reading

http://www-users.cs.york.ac.uk/~ndm/uniplate

5.3 Processing Haskell

5.3.1 hint

Report by:	Daniel Gorin
Status:	active
Current release:	0.2

This library defines a Haskell Interpreter monad. It allows to load Haskell modules, browse them, type-check and evaluate strings with Haskell expressions, and even coerce them into values. The operations are thread-safe and type-safe (even the coercion of expressions to values).

It may be useful for those who need GHCi-like functionality in their programs but do not want to mess with the GHC-API innards. Additionally, unlike the latter, hint provides an API that is consistent across GHC versions.

Works with GHC 6.6.x and 6.8.x.

Further reading

The latest stable version can be downloaded from Hackage.

5.3.2 hs-plugins

Report by:	Don Stewart
Status:	maintained

hs-plugins is a library for dynamic loading and runtime compilation of Haskell modules, for Haskell and foreign language applications. It can be used to implement application plugins, hot swapping of modules in running applications, runtime evaluation of Haskell, and enables the use of Haskell as an application extension language.

hs-plugins has been ported to GHC 6.8, and version 1.2 has been released.

Further reading

- Source and documentation can be found at: http://www.cse.unsw.edu.au/~dons/hs-plugins/
- The source repository is available:
  `darcs get`
  http://www.cse.unsw.edu.au/~dons/code/hs-plugins/

5.3.3 hscolour

Report by:	Malcolm Wallace
Status:	stable, maintained

HsColour is a small command-line tool (and Haskell library) that syntax-colourises Haskell source code for multiple output formats. It consists of a token lexer, classification engine, and multiple separate pretty-printers for the different formats. Current supported output formats are ANSI terminal codes, HTML (with

or without CSS), LaTeX, and IRC chat codes. In all cases, the colours and highlight styles (bold, underline, etc.) are configurable. It can additionally place HTML anchors in front of declarations, to be used as the target of links you generate in Haddock (→ 4.2.1) documentation.

HsColour is widely used to make source code in blog entries look more pretty, to generate library documentation on the web, and to improve the readability of GHC's intermediate-code debugging output. The current version is 1.9, adding the mIRC backend, and a few bugfixes.

Further reading

http://www.cs.york.ac.uk/fp/darcs/hscolour

5.4 Parsing and Transforming

5.4.1 pcre-light

Report by:	Don Stewart
Status:	active development

A small, efficient, and portable regex library for Perl 5 compatible regular expressions. The PCRE library is a set of functions that implement regular expression pattern matching using the same syntax and semantics as Perl 5.

Further reading

- Source and documentation can be found on Hackage.
- The source repository is available: `darcs get` http://code.haskell.org/~dons/code/pcre-light/

5.4.2 HStringTemplate

Report by:	Sterling Clover

HStringTemplate is a port of the StringTemplate library to Haskell. StringTemplate is a templating system that enforces strict model-view separation via a Turing-incomplete grammar that nonetheless provides powerful recursive constructs. The library provides template grouping and inheritance, as well as escaping. It is especially suited for rapid and iterative development of web applications. HStringTemplate is currently at release 0.3.1 and is available via Hackage.

Further reading

- http://www.cs.usfca.edu/~parrt/papers/mvc.templates.pdf
- HStringTemplate: http://fmapfixreturn.wordpress.com
- StringTemplate: ttp://www.stringtemplate.org/

5.4.3 CoreErlang

Report by:	Henrique Ferreiro García
Status:	Parses and pretty-prints almost all of Core Erlang

CoreErlang is a Haskell library which consists of a parser and a pretty-printer for the intermediate language used by Erlang. The parser uses the Parsec library, and the pretty-printer was modelled after the corresponding module of the haskell-src package. It also exposes a Syntax module which will be used to implement several Core-to-Core optimisations.

It is not finished yet, but it parses almost all of Core Erlang and the pretty-printer works quite well.

In a very short period of time it will be published on Hackage.

5.4.4 parse-dimacs: A DIMACS CNF Parser

Report by:	Denis Bueno
Status:	Version 1.1

Parse-dimacs is a Parsec parser for a common file format — DIMACS — describing conjunctive normal form (CNF) formulae. CNF formulae are typically used as input to satisfiability solvers.

The parser is available from Hackage: http://hackage.haskell.org/cgi-bin/hackage-scripts/package/parse-dimacs

The next release will concentrate on optimisation, specifically for large CNF formulae. The interface is simple and should be stable.

5.4.5 Graph Parser Combinators in Curry

Report by:	Steffen Mazanek
Status:	research prototype

A graph language can be described with a graph grammar in a manner similar to a string grammar known from the theory of formal languages. In the last HCAR report we have introduced the Haskell library *graph parser combinators*. Therewith, several graph parsers can be implemented quite conveniently.

Unfortunately, it is quite complicated to realize a straightforward and reasonably efficient translation of so-called hyperedge replacement grammars (a context-free graph grammar formalism) to graph parsers. Problems are mainly caused by heavy non-determinism. Therefore, we have reimplemented our library in Curry (→ 3.2.1), a functional-logic programming language.

The Curry implementation provides two main benefits: Grammars can be translated to quite efficient parsers in a schematic way. Furthermore, parsers can be used as generators and for graph completion at the same time.

We exploit these nice properties in the domain of diagram editors. Here, graph grammars are used to define the syntax of visual languages, and graph completion appears to be very beneficial for the realization of powerful content assist. We have connected our framework to the diagram editor generator DiaGen (http://www.unibw.de/inf2/DiaGen).

Our application provides a strong motivation for further research into multi-paradigm declarative languages.

Further reading

http://www.unibw.de/steffen.mazanek/forschung/grappa

5.4.6 The X-SAIGA Project

Report by:	Richard A. Frost
Participants:	Rahmatullah Hafiz, Paul Callaghan
Status:	code available

The goal of the X-SAIGA project is to create algorithms and implementations which enable language processors (recognisers, parsers, interpreters, translators, etc.) to be constructed as modular and efficient embedded eXecutable SpecificAtIons of GrAmmars.

To achieve modularity, we have chosen to base our algorithms on top-down parsing. To accommodate ambiguity, we implement inclusive choice through backtracking search. To achieve polynomial complexity, we use memoisation. We have developed an algorithm which accommodates direct left-recursion using curtailment of search. Indirect left recursion is also accommodated using curtailment together with a test to determine whether previously computed and memoised results may be reused depending on the context in which they were created and the context in which they are being considered for reuse.

The algorithm is described more fully in Frost, R., Hafiz, R., and Callaghan, P. (2007) Modular and Efficient Top-Down Parsing for Ambiguous Left-Recursive Grammars. Proceedings of the 10th International Workshop on Parsing Technologies (IWPT), ACL-SIGPARSE. Pages: 109 – 120, June 2007, Prague. (http://cs.uwindsor.ca/~hafiz/iwpt-07.pdf)

We have implemented our algorithms, at various stages of their development, in Miranda (up to 2006) and in Haskell (from 2006 onwards). A description of a Haskell implementation of our 2007 algorithm can be found in Frost, R., Hafiz, R., and Callaghan, P. (2008) Parser Combinators for Ambiguous Left-Recursive Grammars. Proceedings of the 10th International Symposium on Practical Aspects of Declarative Languages (PADL), Paul Hudak, David Scott Warren (Eds.): Practical Aspects of Declarative Languages, 10th International Symposium, PADL 2008, San Francisco, CA, USA, January 7–8, 2008. Springer 2008, LNCS 4902, 167–181. (http://cs.uwindsor.ca/~hafiz/PADL_PAPER_FINAL.pdf)

The X-SAIGA website contains more information, links to other publications, proofs of termination and complexity, and Haskell code of the development version. (http://cs.uwindsor.ca/~hafiz/proHome.html)

We are currently extending our algorithm and implementation to accommodate executable specifications of fully-general attribute grammars.

One of our long-term goals is to use the X-SAIGA software to construct natural-language applications as executable specifications of attribute grammars and deploy them on the Public-Domain SpeechWeb, which is a related project of ours that is also funded by the Natural Science and Engineering Research Council of Canada (NSERC). More information on the SpeechWeb project, including details of how to access our prototype Public-Domain SpeechWeb by voice, and how to build and deploy your own speech applications, can be found at http://www.myspeechweb.org.

5.4.7 InterpreterLib

Report by:	Jennifer Streb
Participants:	Garrin Kimmell, Nicolas Frisby, Mark Snyder, Philip Weaver, Perry Alexander
Maintainer:	Garrin Kimmell, Nicolas Frisby
Status:	beta, actively developed

The InterpreterLib library is a collection of modules for constructing composable, monadic interpreters in Haskell. The library provides a collection of functions and type classes that implement semantic algebras in the style of Hutton and Duponcheel. Datatypes for related language constructs are defined as non-recursive functors and composed using a higher-order sum functor. The full AST for a language is the least fixed point of the sum of its constructs' functors. To denote a term in the language, a sum algebra combinator composes algebras for each construct functor into a semantic algebra suitable for the full language, and the catamorphism introduces recursion. Another piece of InterpreterLib is a novel suite of algebra combinators conducive to monadic encapsulation and semantic reuse. The Algebra Compiler, an ancillary preprocessor derived from polytypic programming principles, generates functorial boilerplate Haskell code from minimal specifications of language constructs. As a whole, the InterpreterLib library enables rapid prototyping and simplified maintenance of language processors.

InterpreterLib is available for download at the link provided below. Version 1.0 of InterpreterLib was released in April 2007.

Further reading

http://www.ittc.ku.edu/Projects/SLDG/projects/project-InterpreterLib.htm

Contact

⟨nfrisby@ittc.ku.edu⟩

5.5 Data types and data structures

5.5.1 Data.ByteString

Report by:	Don Stewart
Status:	active development

Data.ByteString provides packed strings (byte arrays held by a ForeignPtr), along with a list interface to these strings. It lets you do extremely fast IO in Haskell; in some cases, even faster than typical C implementations, and much faster than `[Char]`. It uses a flexible "foreign pointer" representation, allowing the transparent use of Haskell or C code to manipulate the strings.

Data.ByteString is written in Haskell98 plus the foreign function interface and cpp. It has been tested successfully with GHC 6.4, 6.6, 6.8, Hugs 2005–2006, and the head version of nhc98.

Bytestring 0.9.1.0 has been released, with full coverage data, an improved testsuite, and some key performance improvements.

Further reading

- Source and documentation can be found at http://www.cse.unsw.edu.au/~dons/fps.html
- The source repository is available:
  `darcs get` http://darcs.haskell.org/bytestring

5.5.2 dlist

Report by:	Don Stewart
Status:	active development

Differences lists: a list-like type supporting O(1) append. This is particularly useful for efficient logging and pretty printing, (e.g., with the Writer monad), where list append quickly becomes too expensive.

Further reading

- Source and documentation can be found on Hackage.
- The source repository is available:
  `darcs get` http://code.haskell.org/~dons/code/dlist/

5.5.3 dimensional

Report by:	Björn Buckwalter
Status:	active, mostly stable

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.

The core of dimensional is stable with additional units being added on an as-needed basis. In addition to the SI system of units, dimensional has experimental support for user-defined dimensions and a proof-of-concept implementation of the CGS system of units. I am also experimenting with forward automatic differentiation and rudimentary linear algebra.

The current release is compatible with GHC 6.6.x and above and can be downloaded from Hackage or the project web site. The primary documentation is the literate Haskell source code, but the wiki on the project web site has a few usage examples to help with getting started.

The Darcs repo has moved to http://code.haskell.org/dimensional.

Further reading

http://dimensional.googlecode.com

5.5.4 Numeric prelude

Report by:	Henning Thielemann
Participants:	Dylan Thurston, Mikael Johansson
Status:	experimental, active development

The hierarchy of numerical type classes is revised and oriented at algebraic structures. Axiomatics for fundamental operations are given as QuickCheck properties, superfluous super-classes like `Show` are removed, semantic and representation-specific operations are separated, the hierarchy of type classes is more fine grained, and identifiers are adapted to mathematical terms.

There are both certain new type classes representing algebraic structures and new types of mathematical objects. Currently supported algebraic structures are

- group (additive),
- ring,
- principal ideal domain,
- field,
- algebraic closures,
- transcendental closures,
- module and vector space,
- normed space,
- lattice,
- differential algebra,
- monoid.

There is also a collection of mathematical object types, which is useful both for applications and testing the class hierarchy. The types are

- lazy Peano number,

- complex number, quaternion,
- residue class,
- fraction,
- partial fraction,
- numbers equipped with physical units in two variants:
  1. dynamically checked units,
  2. statically checked dimension terms (E.g., speed can be expressed by type argument `Mul Length (Recip Time)`. This is overly restrictive but does not require type extensions.)
- fixed point arithmetic with respect to arbitrary bases and numbers of fraction digits,
- infinite precision number in an arbitrary positional system as lazy lists of digits supporting also numbers with terminating representations,
- polynomial, power series, LAURENT series
- root set of a polynomial,
- matrix (basics only),
- algebra, e.g., multi-variate polynomial (basics only),
- permutation group.

Due to Haskell's flexible type system, you can combine all these types, e.g., fractions of polynomials, residue classes of polynomials, complex numbers with physical units, power series with real numbers as coefficients.

Using the revised system requires hiding some of the standard functions provided by Prelude, which is fortunately supported by GHC. The library has basic Cabal support and a growing test-suite of QuickCheck tests for the implemented mathematical objects.

Each data type now resides in a separate module. Cyclic dependencies could be eliminated by fixing some types in class methods. E.g., power exponents became simply Integer instead of Integral, which has also the advantage of reduced type defaulting.

Future plans

Collect more Haskell code related to mathematics, e.g., for linear algebra. Study of alternative numeric type class proposals and common computer algebra systems.

A still unsolved problem arises for residue classes, matrix computations, infinite precision numbers, fixed point numbers, and others. It should be possible to assert statically that the arguments of a function are residue classes with respect to the same divisor, or that they are vectors of the same size. Possible ways out are encoding values in types or local type class instances. The latter one is still neither proposed nor implemented in any Haskell compiler. The modules are implemented in a way to keep all options open. That is, for each number type there is one module implementing the necessary operations which expect the context as a parameter. Then there are several modules which provide different interfaces through type class instances to these operations.

Further reading

http://darcs.haskell.org/numericprelude/

5.5.5 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 (→1.6.5). 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 now others (Alexandra Silva, Joost Visser: PURe.CoddFish project), have also used HList for type-safe database access in Haskell. HList-based Records form the basis of OOHaskell (http://darcs.haskell.org/OOHaskell). The HList library relies on common extensions of Haskell 98.

The HList repository is available via Darcs: http://darcs.haskell.org/HList

The main change since the last report was the addition of a large set of patches by Gwern Branwen ⟨gwern0@gmail.com⟩, to arrange the library within the Data.HList hierarchy, to update the code for GHC 6.8.2 (using the LANGUAGE pragma, eliminating causes of GHC warnings), to build the library with the latest version of Cabal. He also uploaded the library to Hackage. Many thanks to Gwern Branwen.

Further reading

- HList: http://homepages.cwi.nl/~ralf/HList/
- OOHaskell: http://homepages.cwi.nl/~ralf/OOHaskell/

5.5.6 stream-fusion

Report by:	Don Stewart
Status:	active development

Data.List.Stream provides the standard Haskell list data type and api, with an improved fusion sys-

tem, as described in the papers "Stream Fusion" and "Rewriting Haskell Strings". Code written to use the Data.List.Stream library should run faster (or at worst, as fast) as existing list code. A precise, correct reimplementation is a major goal of this project, and Data.List.Stream comes bundled with around 1000 QuickCheck properties, testing against the Haskell98 specification, and the standard library.

The latest version of the stream-fusion package is now available from Hackage.

Further reading

Source and documentation can be found at: http://www.cse.unsw.edu.au/~dons/streams.html

5.5.7 Edison

Report by:	Robert Dockins
Status:	stable, maintained

Edison is a library of purely function data structures for Haskell originally written by Chris Okasaki. Conceptually, it consists of two things:

1. A set of type classes defining the following data structure abstractions: "sequences", "collections", and "associative collections"

2. Multiple concrete implementations of each of the abstractions.

In theory, either component may be used independently of the other.

I took over maintenance of Edison in order to update Edison to use the most current Haskell tools. The following major changes have been made since version 1.1, which was released in 1999.

- Typeclasses updated to use fundeps (by Andrew Bromage)
- Implementation of ternary search tries (by Andrew Bromage)
- Modules renamed to use the hierarchical module extension
- Documentation haddockised
- Source moved to a Darcs repository
- Build system cabalised
- Unit tests integrated into a single driver program which exercises all the concrete implementations shipped with Edison
- Multiple additions to the APIs (mostly the associated collection API)

Edison is currently in maintain-only mode. I do not have the time required to enhance Edison in the ways I would like. If you are interested in working on Edison, do not hesitate to contact me.

The biggest thing that Edison needs is a benchmarking suite. Although Edison currently has an extensive unit test suite for testing correctness, and many of the data structures have proven time bounds, I have no way to evaluate or compare the quantitative performance of data structure implementations in a principled way. Unfortunately, benchmarking data structures in a non-strict language is difficult to do well. If you have an interest or experience in this area, your help would be very much appreciated.

Further reading

http://www.cs.princeton.edu/~rdockins/edison/home/

5.6 Data processing

5.6.1 bytestring-mmap

Report by:	Don Stewart
Status:	active development

This library provides a wrapper to mmap(2), allowing files or devices to be lazily loaded into memory as strict or lazy ByteStrings (→ 5.5.1), using the virtual memory subsystem to do on-demand loading.

Further reading

- Source and documentation can be found on Hackage.
- The source repository is available:
  `darcs get` http://code.haskell.org/~dons/code/bytestring-mmap/

5.6.2 binary

Report by:	Lennart Kolmodin
Participants:	Duncan Coutts, Don Stewart, Binary Strike Team
Status:	active

The Binary Strike Team is pleased to announce yet another release of a new, pure, efficient binary serialisation library.

The "binary" package provides efficient serialisation of Haskell values to and from lazy ByteStrings. ByteStrings constructed this way may then be written to disk, written to the network, or further processed (e.g., stored in memory directly, or compressed in memory with zlib or bzlib).

The binary library has been heavily tuned for performance, particularly for writing speed. Throughput of up to 160M/s has been achieved in practice, and in general speed is on par or better than NewBinary,

with the advantage of a pure interface. Efforts are underway to improve performance still further. Plans are also taking shape for a parser combinator library on top of binary, for bit parsing and foreign structure parsing (e.g., network protocols).

Data.Derive has support for automatically generating Binary instances, allowing to read and write your data structures with little fuzz.

Binary was developed by a team of 8 during the Haskell Hackathon in Oxford 2007, and since then about 15 people have contributed code and many more given feedback and cheerleading on `#haskell` (→ 1.2).

The package is cabalised and available through Hackage.

Further reading

- Homepage: http://code.haskell.org/binary/
- Hackage: http://hackage.haskell.org/cgi-bin/hackage-scripts/package/binary
- Development version: `darcs get -partial` http://code.haskell.org/binary

5.6.3 The Haskell Cryptographic Library

Report by:	Dominic Steinitz

The latest version is 4.1.0. Contributions since the last update include: TEA, BubbleBabble, HMAC, and more flavours of SHA.

The interface to SHA-1 is still different from MD5, and the whole library still needs a rethink. Unfortunately, I still do not have the time to undertake much work on it at the moment and it is not clear when I will have more time. I am still therefore looking for someone to help keeping the repository up-to-date with contributions, re-structuring the library, and managing releases.

This release contains:

- DES
- Blowfish
- AES
- TEA
- BubbleBabble
- Cipher Block Chaining (CBC)
- PKCS#5 and nulls padding
- SHA-1, SHA-2, SHA-224, SHA-256, SHA-384, SHA-512
- HMAC
- MD5
- RSA
- OAEP-based encryption (Bellare-Rogaway)
- Hex utilities
- Support for Word128, Word192 and Word256, and beyond

Further reading

- http://www.haskell.org/crypto
- http://hackage.haskell.org/trac/crypto.

5.6.4 The Haskell ASN.1 Library

Report by:	Dominic Steinitz

The current release remains 0.0.11, which contains functions to handle ASN.1, X.509, PKCS#8, and PKCS#1.5.

This still has a dependency on NewBinary but we now have a way of removing this to use ByteStrings (→ 5.5.1), although the work remains to be done.

The current version handles the Basic Encoding Rules (BER). In addition, even more work (over 400 Darcs patches) has been undertaken on handling the Packed Encoding Rules (PER) using a GADT to represent the Abstract Syntax Tree (we will probably move the BER to use the same AST at some point). Interestingly, this has resulted in us finding a small bug in the ASN.1 specification which we have reported to the ITU. You can download the current working version and try the unit and QuickCheck property tests for PER. These are not yet built by Cabal.

This release supports:

- X.509 identity certificates
- X.509 attribute certificates
- PKCS#8 private keys
- PKCS#1 version 1.5

Further reading

http://haskell.org/asn1.

5.6.5 2LT: Two-Level Transformation

Report by:	Tiago Miguel Laureano Alves
Participants:	Joost Visser, Pablo Berdaguer, Alcino Cunha, José Nuno Oliveira, Hugo Pacheco
Status:	active

A two-level data transformation consists of a type-level transformation of a data format coupled with value-level transformations of data instances corresponding to that format. Examples of two-level data transformations include XML schema evolution coupled with document migration, and data mappings used for interoperability and persistence.

In the 2LT project, support for two-level transformations is being developed using Haskell, relying in particular on generalised abstract data types (GADTs). Currently, the 2LT package offers:

- A library of two-level transformation combinators. These combinators are used to compose transformation systems which, when applied to an input type,

produce an output type together with the conversion functions that mediate between input and output types.

- Front-ends for VDM-SL, XML, and SQL. These front-ends support (i) reading a schema, (ii) applying a two-level transformation system to produce a new schema, (iii) converting a document/database corresponding to the input schema to a document/database corresponding to the output schema, and *vice versa*.

- A combinator library for transformation of point-free and structure-shy functions. These combinators are used to compose transformation systems for optimisation of conversion functions, and for migration of queries through two-level transformations. Independently of two-level transformation, the combinators can be used to specialise structure-shy programs (such as XPath queries and strategic functions) to structure-sensitive point-free form, and *vice versa*.

- Support for schema constraints using point-free expressions. Constraints present in the initial schema are preserved during the transformation process and new constraints are added in specific transformations to ensure semantic preservation. Constraints can be simplified using the already existent library for transformation of point-free functions.

The various sets of transformation combinators are reminiscent of the combinators of Strafunski and the Scrap-your-Boilerplate approach to generic functional programming.

A release of 2LT is available from the project URL.

Recently, the 2LT project has been migrated to Google Code. New functionality is planned, such as elaboration of the front-ends and the creation of a web interface.

Further reading

- Project URL: http://2lt.googlecode.com
- Alcino Cunha, José Nuno Oliveira, Joost Visser. *Type-safe Two-level Data Transformation*. Formal Methods 2006.
- Alcino Cunha, Joost Visser. Strongly Typed Rewriting For Coupled Software Transformation. RULE 2006.
- Pablo Berdaguer, Alcino Cunha, Hugo Pacheco, Joost Visser. *Coupled Schema Transformation and Data Conversion For XML and SQL*. PADL 2007.
- Alcino Cunha and Joost Visser. *Transformation of Structure-Shy Programs, Applied to XPath Queries and Strategic Functions*. PEPM 2007.
- Tiago L. Alves, Paulo Silva and Joost Visser. *Constraint-aware Schema Transformation*. Draft, 2007.

5.7 Types for Safety and Reasoning

5.7.1 Takusen

Report by:	Alistair Bayley
Participants:	Oleg Kiselyov
Status:	active development

Takusen is a library for accessing DBMS's. Like HSQL, we support arbitrary SQL statements (currently strings, extensible to anything that can be converted to a string).

Takusen's "unique selling-point" is safety and efficiency. We statically ensure that all acquired database resources such as cursors, connection, and statement handles are released, exactly once, at predictable times. Takusen can avoid loading the whole result set in memory and so can handle queries returning millions of rows, in constant space. Takusen also supports automatic marshaling and unmarshaling of results and query parameters. These benefits come from the design of query result processing around a left-fold enumerator.

Currently we fully support Oracle, Sqlite, and PostgreSQL, and ODBC support exists but is not fully tested.

Since the last report we have:

- made bug fixes and enhancements to the ODBC backend

- improved the installation process so that we can build Haddock docs with Cabal

- added an inquire function (and EnvInquiry class), which lets us call arbitrary DBMS functions. This provides a general interface to invoke arbitrary backend interrogation functions without compromising the framework guarantees. This is demonstrated by LastInsertRowid in the Sqlite backend.

- re-exported a bunch of types from Database.InternalEnumerator, which should help people wanting to write type sigs for database functions.

Future plans

- complete ODBC interface.
- Large object support.
- MS SQL Server and Sybase interfaces, via FreeTDS.

Further reading

- `darcs get` http://darcs.haskell.org/takusen/
- browse docs: http://darcs.haskell.org/takusen/doc/html (see Database.Enumerator for Usage instructions and examples)

5.7.2 Session Types for Haskell

Report by:	Matthew Sackman
Status:	beta; active development

Session Types provide a way to express communication protocols. They specify, for a bi-directional channel, who says what, in what order, and to whom. Looping and branching structures are supported. Thus a session type adds a type to a communication channel, ensuring that use of the channel is safe (i.e., when one party speaks, the others listen, and that the type of the value sent is the type of the value expected to be received). Thus, Session Types offer temporal information which is absent from all other concurrency techniques.

The focus of the library is on the communication between threads. However, work is progressing on supporting fully distributed operation. The library supports forking new processes with channels; creating new channels between existing processes; the communication of process IDs; the communication of channels (higher-order channels or delegation); subtyping of Pids; and some initial work on real distributed operation over Handles.

Current development is rapid and is focusing on building up a strong suite of examples and networked operation. Recent features have added support for higher-order channels and a new DSL for specifying Session Types (which supports lexical scoping and is composable).

If you are doing any multi-threaded development in Haskell and find the properties and simplicity of message passing concurrency attractive, then I strongly encourage you to take a look at Session Types.

Further reading

- Project homepage: http://wellquite.org/sessions/
- Tutorial: http://wellquite.org/sessions/tutorial_1.html

5.7.3 Category Extras — Comonad Transformers and Bird-Meertens combinators

Report by:	Edward Kmett
Status:	experimental

Haskell has derived a lot of benefits from the tools it has borrowed from category theory, such as functors, monads, and arrows. However, there are a lot more tools out there. For instance, comonads have been barely touched by Haskell programmers. This library attempts to collect many of the more interesting bits of category theory and constructive algorithmics in one place and generalise or dualize previous results where possible.

The library includes:

- A set of comonad transformers in the spirit of the monad transformer library, including the categorical duals of the Reader, State, and Writer monads and monad transformers: the Product, Context, and Supply comonads and comonad transformers, respectively.
- An expanded set of "Bananas, Lenses, and Barbed Wire" for constructive algorithmics, including new generalised hylomorphisms and combinators for building up the distributive laws needed to use them
- Left and right Kan extensions
- Generalisations of standard library functions such as zip and unzip
- Free monads and cofree comonads with free monad coproducts and cofree comonad coproducts
- Ideal monads and their previously unpublished dual, coideal comonads, with ideal monad coproducts and cofree comonad products
- Higher-order functors, adjunctions, monads, and comonads that work over functors and map natural transformations
- Indexed (co)monads, including the well-known implementations for indexed state and delimited continuations
- Hyperfunctions
- Multiple functor composition operators to support (co)monad/(co)pointed functor composition and adjunction based (co)monads without ambiguity

Future plans

- Zippered comonadic automata
- A suite of (bi)functor type-level combinators inspired by similar term-level combinators such as On, Ap, and Join to help see the connections between different (co)monads and to simplify the taking of type-level derivatives.
- A "cofib" example suite to demonstrate programming with comonads.
- Better documentation

Further reading

http://comonad.com/haskell/category-extras/

5.7.4 IOSpec

Report by:	Wouter Swierstra
Status:	active development

The IOSpec library provides a pure specification of several functions in the IO monad. This may be of interest to anyone who wants to debug, reason about, analyse, or test impure code.

The IOSpec library is essentially a drop-in replacement for several other modules, such as Data.IORef and Control.Concurrent. Once you are satisfied that your functions are reasonably well-behaved with respect to the pure specification, you can drop the IOSpec import in favour of the "real" IO modules. The ideas underlying the previous version are described by a recent Haskell Workshop paper.

The latest version, however, supports several exciting new features. Besides providing a pure specification of STM, it allows you to build your own pure IO monad à la carte — allowing you to be explicit about which effects your program is using.

In the next major release, I would like to incorporate efficiency improvements suggested by Janis Voigtländer and allow the extraction of an IO computation from its pure counterpart.

If you use IOSpec for anything useful at all, I would love to hear from you.

Further reading

http://www.cs.nott.ac.uk/~wss/repos/IOSpec/

5.8 User interfaces

5.8.1 Gtk2Hs

Report by:	Axel Simon
Participants:	Duncan Coutts
Status:	beta, actively developed

Gtk2Hs is a GUI Library for Haskell based on Gtk+. 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 and, of course, various desktops on Linux, Solaris, and FreeBSD. Gtk+ and Gtk2Hs also support Mac OS X (it currently uses the X11 server, but a native port is in progress — see below).

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
- support for the Glade visual GUI builder
- bindings to some Gnome extensions: GConf, a source code editor widget, and a widget that embeds the Mozilla/Firefox rendering engine
- an easy-to-use installer for Windows
- packages for Fedora, Gentoo (→ 2.6.1), Debian, and FreeBSD

The Gtk2Hs library is in a usable state and many parts of the API can be considered stable. We received very positive feedback on tutorials which were contributed by various people (→ 1.6.4). In fact, although the maintainers of Gtk2Hs manage to improve the library itself, we would greatly benefit from more outside help, for instance, somebody who can create releases and fix (configuration) bugs. The reason that we have not released in the last six months is simply lack of time (and access to platforms), even though the list and tree widgets have been complete since January. We furthermore implemented more functions in Pango, the font rendering engine. More frequent releases are needed to get these improvements to people who rely on binary installers.

Our further development goals are as follows: In the medium term we hope to support the new features in Gtk+ 2.12 and to improve the signals API. In the longer term we hope to modularise Gtk2Hs and enable it to be built and distributed with Cabal and Hackage. A promising recent development is that Gtk+'s native (non-X11) backend for Mac OS X has got to the point where Gtk2Hs can be built against it and most of the demo programs work. It would be great if we could find somebody who can create a Mac OS installer that bundles Gtk2Hs with the native Aqua port of Gtk.

Further reading

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

5.8.2 Grapefruit — A declarative GUI and graphics library

Report by:	Wolfgang Jeltsch
Participants:	Matthias Reisner
Status:	provisional

Grapefruit is a library for creating graphical user interfaces and animated graphics in a declarative way.

Fundamental to Grapefruit is the notion of signal. A signal denotes either a time-varying value (the continuous case) or a sequence of values assigned to discrete points in time (the discrete case). Signals can be constructed in a purely functional manner.

User interfaces are described as systems of interconnected components which communicate via signals. To build such systems, the methods from the Arrow and ArrowLoop classes are used. For describing animated graphics, a special signal type exists.

Grapefruit also provides list signals. A list signal is a list-valued signal which can be updated incrementally and thus efficiently. In addition, a list signal associates an identity with each element, so that moving

an element within the list can be distinguished from removing the element and adding it again. List signals can be used to describe dynamic user interfaces, i.e., user interfaces with a changing set of components and changing order of components.

Grapefruit descriptions of user interfaces and animations always cover their complete lifetime. No explicit event handler registrations and no explicit recalculations of values are necessary. This is in line with the declarative nature of Haskell because it stresses the behaviour of GUIs and animations instead of how this behaviour is achieved. Internally, though, Grapefruit is implemented efficiently using a common event dispatching and handling mechanism.

Grapefruit is currently based on Gtk2Hs (→ 5.8.1) and HOpenGL, but implementations on top of other GUI and graphics libraries are possible. The aim is to provide alternative implementations based on different GUI toolkits, so that a single application is able to integrate itself into multiple desktop environments.

Further reading

http://haskell.org/haskellwiki/Grapefruit

5.8.3 Shellac

Report by:	Robert Dockins
Status:	beta, maintained

Shellac is a framework for building read-eval-print style shells. Shells are created by declaratively defining a set of shell commands and an evaluation function. Shellac supports multiple shell backends, including a "basic" backend, which uses only Haskell IO primitives, and a fully featured "readline" backend based on the Haskell readline bindings found in the standard libraries.

This library attempts to allow users to write shells in a declarative way and still enjoy the advanced features that may be available from a powerful line editing package like readline.

Shellac is available from Hackage, as is the related Shellac-readline package.

Shellac has been successfully used by several independent projects, and the API is now fairly stable. I will likely be releasing an officially "stable" version in the not-too-distant future. I anticipate few changes from the current version.

Further reading

http://www.cs.princeton.edu/~rdockins/shellac/home

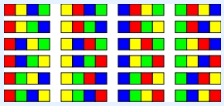
5.9 (Multi-)Media

5.9.1 diagrams

Report by:	Brent Yorgey
Status:	active development

The diagrams library provides an embedded domain-specific language for creating simple pictures and diagrams, built on top of the Cairo rendering engine. Values of type `Diagram` are built up in a compositional style from various primitives and combinators, and can be rendered to a physical medium, such as a file in PNG, PS, PDF, or SVG format.

For example, consider the following diagram to illustrate the 24 permutations of four objects:



The diagrams library was used to create this diagram with very little effort (about ten lines of Haskell, including the code to actually generate permutations). The source code for this diagram, as well as other examples and further resources, can be found at http://code.haskell.org/diagrams/.

The library is still in its infancy and is under active development. New contributors and testers are welcome! In particular, this is an ideal project for anyone relatively new to Haskell who is seeking to contribute to a project as a way of becoming more comfortable with the language.

Further reading

- http://code.haskell.org/diagrams/
- http://byorgey.wordpress.com/2008/04/30/new-haskell-diagrams-library/

5.9.2 YampaSynth (previously: Programming of Modular Synthesisers)

Report by:	George Giorgidze
Status:	Experimental

YampaSynth is a purely functional framework for programming modular synthesisers in Haskell using Yampa, a domain specific language embedded in Haskell for programming hybrid systems. A synthesiser, be it a hardware instrument or a pure software implementation, as here, is said to be modular if it provides sound-generating and sound-shaping components that can be interconnected in arbitrary ways.

Basic sound-generating and sound-shaping modules have been implemented, e.g., oscillator, amplifier,

mixer, envelope generator, filter, etc. These modules are used to develop example applications:

- **yampasynth-wav** is an application which synthesises MIDI music and writes the result into a WAVE audio file.
- **yampasynth-openal** is an application which synthesises MIDI music and sends audio data in real-time to a sound card. We use a Haskell binding of the OpenAL library as an interface to audio hardware.
- **yampasynth-gtk** is an application with a simple graphical user interface that allows you to play music with various instruments in real-time using the keyboard of your computer. We use a Haskell binding of the GTK library for GUI programming, and a Haskell binding of the OpenAL library as an interface to audio hardware.

The source code, together with example applications, has been cabalised and is available under the BSD3 license.

Future plans

We would like to see a richer collection of sound-generating and sound-shaping modules in the framework, and complete implementation of MIDI, SoundFont, and related standards. However, one might find some other interesting continuation of the work. We are open for suggestions and would be happy if someone wishes to collaborate.

Further reading

- Related papers, slides, demos, and talks are available from my homepage
- YampaSynth Cabal package on Hackage
- HCodecs is a supporting library which provides functions to read, write, and manipulate MIDI, WAVE, and SoundFont2 multimedia files. It is written entirely in Haskell.

5.9.3 Haskore revision

Report by:	Henning Thielemann
Participants:	Paul Hudak
Status:	experimental, active development

Haskore is a Haskell library originally written by Paul Hudak that allows music composition within Haskell, i.e., without the need of a custom music programming language. This collaborative project aims at improving consistency, adding extensions, revising design decisions, and fixing bugs. Specific improvements include:

1. Basic Cabal support.
2. The `Music` data type has been generalised in the style of Hudak's "polymorphic temporal media."
3. The `Music` data type has been made abstract by providing functions that operate on it.
4. The notion of instruments is now very general. There are simple predefined instances of the `Music` data type, where instruments are identified by Strings or General MIDI instruments, but any other custom type is possible, including types with instrument specific parameters.
5. Support for CSound orchestra files has been improved and extended, thus allowing instrument design in a signal-processing manner using Haskell, including feedback and signal processors with multiple outputs.
6. Support for the software synthesiser SuperCollider both in real-time and non-real-time mode through the Haskell interface by Rohan Drape.
7. Support for conversion between MIDI and Haskore representation of Music. The MIDI file management has been moved to a separate package http://darcs.haskell.org/midi/. Also, a package for real-time input and output of MIDI events through ALSA is now available: http://darcs.haskell.org/alsa-midi/.
8. A package for lists of events with time information has been factored out, as well as a package for non-negative numbers, which occur as time differences in event lists.
9. The AutoTrack project has been adapted and included.
10. Support for infinite `Music` objects is improved. CSound may be fed with infinite music data through a pipe, and an audio file player like Sox can be fed with an audio stream entirely rendered in Haskell. (See Audio Signal Processing project (→ 6.4.1).)
11. The test suite is based on QuickCheck and HUnit.

Future plans

- Allow modulation of instruments similar to the controllers in the MIDI system. We are currently overhauling the design, such that effects on the music level and effects on the back-end level (MIDI, CSound, SuperCollider, Haskell-Synthesizer) are cleanly separated.
- Split into a core package and add-ons, as soon as Cabal supports that.
- Generate note sheets, say, via Lilypond.
- Connect to other Haskore related projects.
- Microtonal music.

Further reading

- http://www.haskell.org/haskellwiki/Haskore
- http://darcs.haskell.org/haskore/

5.10 Web and XML programming

5.10.1 hvac

Report by:	Sterling Clover

HVAC (Http View and Controller) is a lightweight web application framework based on FastCGI and STM, with an emphasis on concise declarative code and concurrent atomic and transactional logic. The unique feature of hvac is that each request is processed within a single logical transaction that bridges STM, database, and filesystem access. Programs written using hvac are therefore terse and easy to reason about, and should be able to be written largely with applicative combinators alone. Also provided are simply deployed caching combinators with static guarantees as to their properties, as well as a sophisticated library of controller combinators for RESTful programming and a system for typed, composable validation. Future work includes a strong test suite to verify hvac's atomic properties and a strongly statically typed database DSL. An official first release is forthcoming shortly.

Further reading

- http://community.haskell.org/~sclv/hvac/html_docs/~hvac/
- http://fmapfixreturn.wordpress.com

5.10.2 Haskell XML Toolbox

Report by:	Uwe Schmidt
Participants:	Christian Uhlig
Status:	seventh major release (current release: 8.0.0)

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 (→ 5.10.3) 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.

Features

- Validating XML parser
- Very liberal HTML parser
- Lightweight lazy parser for XML/HTML based on Tagsoup (→ 5.10.4)
- Easy de-/serialisation 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
- Native Haskell support of HTTP 1.1 and FILE protocol
- HTTP and access via other protocols via external program curl
- Tested with W3C XML validation suite
- Example programs
- Relax NG schema validator
- An HXT Cookbook for using the toolbox and the arrow interface
- Basic XSLT support
- Darcs repository with current development version (8.0.1) under http://darcs.fh-wedel.de/hxt

Current Work

In a master student's project done by Christian Uhlig, the development of a web server called Janus has been finished. The title is *A Dynamic Webserver with Servlet Functionality in Haskell Representing all Internal Data by Means of XML*. HXT has been used for processing all internal data of this web server. The Janus server is highly configurable and can be used not only as HTTP server, but for various other server-like tasks. This server is used and will be further developed and extended within another project called Holumbus (→ 6.3.1), a framework for developing specialised search engines. The Janus system is available via http://darcs.fh-wedel.de/janus. Current activity consists of testing, example applications, demos, and documentation. An application of Janus and Holumbus is the Hayoo! search engine (http://holumbus.fh-wedel.de/hayoo/).

Further reading

The Haskell XML Toolbox Web page (http://www.fh-wedel.de/~si/HXmlToolbox/index.html) includes downloads, online API documentation, a cookbook with nontrivial examples of XML processing using arrows and RDF documents, and master theses describing the design of the toolbox, the DTD validator, the arrow based Relax NG validator, and the XSLT system. A getting started

tutorial about HXT is available in the Haskell Wiki (http://www.haskell.org/haskellwiki/HXT).

5.10.3 HaXml

Report by:	Malcolm Wallace
Status:	stable, maintained

HaXml provides many facilities for using XML from Haskell. The public stable release is 1.13.3, with support for building via Cabal for ghc-6.8.x.

The development version (currently at 1.19, also available through a Darcs repository) includes a much-requested lazy parser and a SAX-like streaming parser. Some minor work still remains to tidy things up before the development version is tagged and released as stable.

The lazy parser combinators used by HaXml now live in a separate library package called polyparse.

Further reading

- http://haskell.org/HaXml
- http://www.cs.york.ac.uk/fp/HaXml-devel
- `darcs get` http://darcs.haskell.org/packages/HaXml
- http://www.cs.york.ac.uk/fp/polyparse

5.10.4 tagsoup

Report by:	Neil Mitchell

TagSoup is a library for extracting information out of unstructured HTML code, sometimes known as tagsoup. The HTML does not have to be well formed, or render properly within any particular framework. This library is for situations where the author of the HTML is not cooperating with the person trying to extract the information, but is also not trying to hide the information.

The library provides a basic data type for a list of unstructured tags, a parser to convert HTML into this tag type, and useful functions and combinators for finding and extracting information. The library has seen real use in an application to give Hackage (→ 5.1) listings, and is used in the next version of Hoogle (→ 4.4.1).

Work continues on the API of tagsoup, and the implementation. Lots of people have made use of tagsoup in their applications, generating lots of valuable feedback. A new version of tagsoup is imminent.

Further reading

http://www-users.cs.york.ac.uk/~ndm/tagsoup

5.10.5 WASH/CGI — Web Authoring System for Haskell

Report by:	Peter Thiemann

WASH/CGI is an embedded DSL (read: a Haskell library) for server-side Web scripting based on the purely functional programming language Haskell. Its implementation is based on the portable common gateway interface (CGI) supported by virtually all Web servers. WASH/CGI offers a unique and fully-typed approach to Web scripting. It offers the following features:

- complete interactive server-side script in one program
- a monadic, type-safe interface to generating XHTML output
- type-safe compositional approach to specifying form elements; callback-style programming interface for forms
- type-safe interfaces to state with different scopes: interaction, persistent client-side (cookie-style), persistent server-side
- high-level API for reading, writing, and sending email
- documented preprocessor for translating markup in syntax close to XHTML syntax into WASH/HTML

Completed items are:

- fully cabalised
- WASH server pages with a modified version of Simon Marlow's `hws` web server; the current prototype supports dynamic compilation and loading of WASH source (via Don Stewart's hs-plugins (→ 5.3.2)) as well as the implementation of a session as a continually running server thread
- Transactional interface to server-side variables and to databases. The interface is inspired by the work on STM (software transactional memory), but modified to be useful in the context of web applications. The interface relies on John Goerzens hdbc package and its PostgreSQL driver.

Current work includes:

- improvement of the database interface
- authentication interface
- user manual (still in the early stages)

Further reading

The WASH Webpage (http://www.informatik.uni-freiburg.de/~thiemann/WASH/) includes examples, a tutorial, a draft user manual, and papers about the implementation.

5.11 System

5.11.1 hinotify

Report by:	Lennart Kolmodin
Status:	alive

"hinotify" is a simple Haskell wrapper for the Linux kernel's inotify mechanism. inotify allows applications to watch file changes, since Linux kernel 2.6.13. You can for example use it to do a proper locking procedure

on a set of files, or keep your application up do date on a directory of files in a fast and clean way.

As file and directory notification is available for many operating systems, upcoming work will include to try to find a common API that could be shared for all platforms. Most recent work has been to see what is possible to do under Microsoft Windows, and finding a suitable API for both platforms. This has been a joint work with Niklas Broberg. We are still looking for contributors to *BSD and Mac OS X. If you are interested, contact us.

Further reading

- Development version: `darcs get` http://www.haskell.org/~kolmodin/code/hinotify/
- Latest released version: http://www.haskell.org/~kolmodin/code/hinotify/download/
- Documentation: http://www.haskell.org/~kolmodin/code/hinotify/docs/api
- inotify: http://www.kernel.org/pub/linux/kernel/people/rml/inotify/

5.11.2 hspread

Report by:	Andrea Vezzosi
Participants:	Jeff Muller
Status:	active

hspread is a client library for the Spread toolkit. It is fully implemented in Haskell using the binary package (→ 5.6.2) for fast parsing of network packets. Its aim is to make it easier to implement correct distributed applications by taking advantage of the guarantees granted by Spread, such as reliable and total ordered messages, and it supports the most recent version of the protocol.

There is interest in further developing a higher level framework for Haskell distributed programming by extending the protocol if necessary.

Further reading

- Hackage: http://hackage.haskell.org/cgi-bin/hackage-scripts/package/hspread
- Development version: `darcs get` http://happs.org/repo/hspread
- Spread homepage: http://www.spread.org

5.11.3 Harpy

Report by:	Dirk Kleeblatt
Participants:	Martin Grabmüller
Status:	experimental

Harpy is a library for run-time code generation of IA-32 machine code. It provides not only a low level interface to code generation operations, but also a convenient domain specific language for machine code fragments, a collection of code generation combinators, and a disassembler. We use it in two independent (unpublished) projects: On the one hand, we are implementing a just-in-time compiler for functional programs, on the other hand, we use it to implement an efficient type checker for a dependently typed language. It might be useful in other domains where specialised code generated at run-time can improve performance.

Harpy's implementation makes use of the foreign function interface, but only contains functions written in Haskell. Moreover, it has some uses of other interesting Haskell extensions, as for example multi-parameter type classes to provide an in-line assembly language, and Template Haskell to generate stub functions to call run-time generated code. The disassembler uses Parsec to parse the instruction stream.

We intend to implement supporting operations for garbage collectors cooperating with run-time generated code.

A second release is forthcoming, featuring improvements in the memory management, better floating point instruction support, and named labels that are shown in the disassembler output.

Further reading

http://uebb.cs.tu-berlin.de/harpy/

6 Applications and Projects

6.1 For the Masses

6.1.1 Darcs

Report by:	Jason Dagit
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.

The year 2008 has seen some major milestones reached in the development of Darcs: years of development of Darcs-2 culminated in a recent release of Darcs 2.0.0. Darcs 2.0.0.0 introduces a number of new features and bug fixes big and small. The most notable changes include fixing the "conflict bug", supporting a new repository format with refined semantics, adding hashed repositories for increased speed and robustness, and a global cache for faster downloading of patches. In the 1.x branch of Darcs, since the last HCAR, support for GHC 6.8.x was added.

Darcs has also seen a revamp of its infrastructure. User visible changes include a consolidation of the stable and unstable development branches, and separation of functions for the two mailing lists. Future work for Darcs will centre around building interesting new features using the new repository format, continued optimising and code restructuring, improving testing of Darcs and adding buildbots, and completing transition to "type witnesses" to ensure provably correct patch manipulation.

Patches great and small are heartily welcomed!

Darcs is free software licensed under the GNU GPL.

Further reading

http://darcs.net

6.1.2 xmonad

Report by:	Don Stewart
Status:	active development

xmonad is a tiling window manager for X. Windows are arranged automatically to tile the screen without gaps or overlap, maximising 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.

The new release 0.7 of xmonad added full support for the GNOME and KDE desktops, and adoption continues to grow, with binary packages of xmonad available for all major 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⟩

6.2 Education

6.2.1 Exercise Assistants

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

At the Open Universiteit Nederland we are building a collection of tools that support students in solving exercises incrementally by checking intermediate steps. All our tools are completely written in Haskell. The distinguishing feature of our tools is the detailed feedback that they provide, on several levels. For example, we have an online exercise assistant that helps to rewrite logical expressions into disjunctive normal form. Students get instant feedback when solving an exercise, and can ask for a hint at any point in the derivation. Other areas covered by our tools are solving linear equations, reducing matrices to echelon normal form, and basic operations on fractions.

The simplest kind of error to deal with are the syntactical errors, for which we use an error correcting parser combinator library. For each exercise domain, we have formulated a set of rewrite rules, as well as a number of unsound (or buggy) rules to catch common mistakes. With these rules we can check all intermediate steps submitted by the user. We also defined

strategies for solving the exercises. A strategy dictates in which order the rules have to be applied to reach the solution, and such a strategy takes the form of a context-free grammar. Strategies are a powerful means to report helpful and informative feedback.

We are offering our tools and our strategies as services to other e-learning tools and environments, such as MathDox and LeActiveMath. For the near future, we have scheduled sessions with students from our university to validate our approach, and to collect information about the usability of our tools. We plan to use generic programming techniques to support exercises from many more, different domains.

An online prototype version for rewriting logical expressions is available and can be accessed from our project page.

Further reading

- http://ideas.cs.uu.nl/trac
- Strategies for exercises. Bastiaan Heeren, Johan Jeuring, Arthur van Leeuwen, and Alex Gerdes. Technical report UU-CS-2008-001, Utrecht University, 2008. To appear in: International Conference on Mathematical Knowledge Management (MKM'08).

6.2.2 Holmes, plagiarism detection for Haskell

Report by:	Jurriaan Hage
Participants:	Brian Vermeer

Years ago, Jurriaan Hage developed Marble to detect plagiarism among Java programs. Marble was written in Perl, takes just 660 lines of code and comments, and does the job well. The techniques used there, however, do not work well for Haskell, which is why a master thesis project was started, starring Brian Vermeer as the master student, to see if we can come up with a working system to discover plagiarism among Haskell programs. We are fortunate to have a large group of students each year that try their hand at our functional programming course (120-130 per year), and we have all the loggings of Helium that we hope can help us tell whether the system finds enough plagiarism cases. The basic idea is to implement as many metrics as possible, and to see, empirically, which combination of metrics scores well enough for our purposes. The implementation will be made in Haskell. One of the things that we are particularly keen about, is to make sure that for assignments in which students are given a large part of the solution and they only need to fill in the missing parts, we still obtain good results.

6.2.3 Geordi IRC C++ eval bot

Report by:	Eelis van der Weegen
Status:	mature

Geordi is an IRC bot that compiles and (optionally) runs C++ code snippets. It has proved to be a very useful tool when teaching and discussing C++ on IRC. It is written in Haskell, and, being deployed on C++ channels at most of the big IRC networks, has the sneaky side-effect of getting some C++'ers interested in Haskell ;-).

Snapshots and Darcs repository can be found at the homepage.

Further reading

http://www.eelis.net/geordi/

6.2.4 Lambda Shell

Report by:	Robert Dockins
Status:	beta, maintained

The Lambda Shell is a feature-rich shell environment and command-line tool for evaluating terms of the pure, untyped lambda calculus. The Lambda Shell builds on the shell creation framework Shellac (→ 5.8.3), and showcases most of Shellac's features.

Features of the Lambda Shell include:

- Evaluate lambda terms directly from the shell prompt using normal or applicative order. In normal order, one can evaluate to normal form, head normal form, or weak head normal form.
- Define aliases for lambda terms using a top level, non-recursive "let" construct.
- Show traces of term evaluation, or dump the trace to a file.
- Count the number of reductions when evaluating terms.
- Test two lambda terms for beta-equivalence (that is; if two terms, when evaluated to normal form, are alpha equivalent).
- Programs can be entered from the command line (using the `-e` option) or piped into stdin (using the `-s` option).
- Perform continuation passing style (CPS) transforms on terms before evaluation using the double-bracket syntax, e.g., "[[five]]".

The Lambda Shell was written as a showcase and textbook example for how to use the Shellac shell-creation library. However, it can also be used to gain a better understanding of the pure lambda calculus.

Further reading

- http://www.cs.princeton.edu/~rdockins/lambda/home
- http://www.cs.princeton.edu/~rdockins/shellac/home

6.2.5 INblobs – Interaction Nets interpreter

Report by:	Miguel Vilaca
Participants:	Daniel Mendes
Status:	active, maintained
Portability:	portable (depends on wxHaskell)

INblobs is an editor and interpreter for Interaction Nets — a graph-rewriting formalism introduced by Lafont, inspired by Proof-nets for Multiplicative Linear Logic.

INblobs is built on top of the front-end Blobs from Arjan van IJzendoorn, Martijn Schrage, and Malcolm Wallace.

The tool is being developed using the repository system Darcs.

New features

- easier implementation of new reduction strategies
- automatic transformation of lambda terms into interaction nets
- generation of textual descriptions allowing the use of INblobs as an editor/frontend for textual IN compilers
- allow creation of properties' checks
- *Valid IN System* check
- minor changes for better usability

Current Work

A new plugin that will allow INblobs to compile a textual functional program into Interaction Nets is being developed.

Further reading

- Homepage: http://haskell.di.uminho.pt/jmvilaca/INblobs/
- Blobs: http://www.cs.york.ac.uk/fp/darcs/Blobs

6.3 Data Access and Visualisation

6.3.1 Holumbus Search Engine Framework

Report by:	Uwe Schmidt
Participants:	Timo B. Hübel, Sebastian Schlatt, Stefan Schmidt
Status:	first beta release

Description

The Holumbus framework consists of a set of modules and tools for creating fast, flexible, and highly customisable 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 web interface for Hayoo! is implemented with the Janus web server, written in Haskell and based on HXT (→ 5.10.2).

Features

- Highly configurable crawler module for flexible indexing of structured data
- Customisable index structure for an effective search
- *find as you type* search
- Suggestions
- Fuzzy queries
- Customisable result ranking
- Index structure designed for distributed search
- Darcs repository with current development version under http://darcs.fh-wedel.de/holumbus

Current Work

Currently the indexer module will further be developed and extended, such that the configuration about the relevant information especially in web pages to be indexed becomes simple and easy. During this activity, new use cases of the framework will be implemented.

In another Master Thesis the distributed query evaluation in a network of machines is tackled. Here we will try to adopt ideas from the Google *map–reduce* approach for Holumbus. The aim is to develop a generally applicable map–reduce like framework and take the Holumbus search and index manipulation as a serious test case.

Further reading

The Holumbus web page (http://holumbus.fh-wedel.de/) includes downloads, Darcs web interface, current status, requirements, and documentation. Timo Hübel'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.

6.3.2 Top Writer

Report by:	Jon Strait
Status:	experimental, active development

Top Writer is a web application for technical writers to easily edit and assemble topic-oriented user guides and other high level references works. Application users edit within a structured framework, using meaningful application elements to chunk and contain content. Users can extend the application with their own elements and rules, if needed. Delivery of content is meant to be multi-format, with each format having separate templating rules.

The server portion of the application is coded in

Haskell using the HAppS framework, and the client web browser portion uses the JQuery Javascript toolkit.

Future plans

Currently, the focus for delivering output is on generated HTML, but plans are also to generate PDF and any other format that is reasonable.

Further reading

http://www.moonloop.net/topwriter

6.3.3 tiddlyisar

Report by:	Slawomir Kolodynski
Status:	under development

tiddlyisar is a tool for generating TiddlyWiki renderings of IsarMathLib source. IsarMathLib is a library of mathematical proofs formally verified by the Isabelle/ZF theorem proving environment. The tiddlywiki tool parses IsarMathLib source and generates TiddlyWiki markup text. The generated view features jsMath based mathematical symbols, cross referenced theorems, and structured proofs expanded on request. The rendering can be viewed on the Tiddly Formal Math site. tiddlyisar will be included in the next release of the IsarMathLib distribution under GPLv3 license.

Further reading

- http://savannah.nongnu.org/projects/isarmathlib
- http://www.cl.cam.ac.uk/research/hvg/Isabelle/
- http://formalmath.tiddlyspot.com

6.3.4 Emping

Report by:	Hans van Thiel

Emping 0.5 has been released. Emping is a (prototype of) a tool for the analysis of multi-variate nominal data. For example, in a table of 8000 mushrooms and 20 attributes, constructed from a field guide, the tool finds which attribute-values determine whether a mushroom is edible or poisonous. But Emping finds not only single factors, but also pairs, triples, and all other combinations which distinguish between the consequent values. Such reduced rules are generalisations of rows in the original table, so r1 could stand for originals a,b,c and r2 for a,b. In that case r2 implies r1 or, conversely, r1 entails r2. The reductions are partially ordered. Emping also finds all such dependencies, including the equivalences where different reductions stand for the same original rules. New in Emping 0.5 is that, thanks to the functional graph library which comes with GHC, these dependencies are now expressed in a Graphviz format and can be shown with a Graphviz reader. Also new is the sort by length of the reduced rules, and of the reduced rules in each equivalence class. This makes the results much more readable. Starting in 0.4, it is now also possible to have blank fields, but this feature has only been summarily tested. The Gtk2Hs based GUI (→ 5.8.1), first introduced in version 0.4, has been improved in Emping 0.5. Data tables, as well as output tables of reduced rules and graph legends, all use the default CSV format of the Open Office Calc spreadsheet.

Further reading

See http://home.telfort.nl/sp969709/emp/empug.html for more, including two white papers and downloads.

6.3.5 SdfMetz

Report by:	Tiago Miguel Laureano Alves
Participants:	Joost Visser
Status:	stable, maintained

SdfMetz supports grammar engineering by calculating grammar metrics and other analyses. Currently it supports four different grammar formalisms (SDF, DMS, Antlr, and Bison) from which it calculates size, complexity, structural, and ambiguity metrics. Output is a textual report or in Comma Separated Value format. The additional analyses implemented are visualisation, showing the non-singleton levels of the grammar, or printing the grammar graph in DOT format. The definition of all except the ambiguity and the NPath metrics were taken from the paper *A metrics suite for grammar based-software* by James F. Power and Brian A. Malloy. The ambiguity metrics were defined by the tool author exploiting specific aspects of SDF grammars, and the NPath metric definition was taken from the paper *NPATH: a measure of execution path complexity and its applications.*

Future plans

Efforts are underway to develop functionalities to compute quality profiles based on histograms. Furthermore, more metrics will be added, and a web-interface is planned.

The tool was initially developed in the context of the IKF-P project (Information Knowledge Fusion, http://ikf.sidereus.pt/) to develop a grammar for ISO VDM-SL.

Further reading

The web site of SdfMetz (http://wiki.di.uminho.pt/wiki/bin/view/PURe/SdfMetz) includes tables of metric values for a series of SDF grammar as computed by SdfMetz. The tool is distributed as part of the UMinho Haskell Libraries and Tools.

6.4 Audio and Graphics

6.4.1 Audio signal processing

Report by:	Henning Thielemann
Status:	experimental, active development

In this project, audio signals are processed using pure Haskell code. The highlights are:

- a basic signal synthesis backend for Haskore (→ 5.9.3),
- experimental structures for filter networks,
- basic audio signal processing, including some hard-coded frequency filters,
- advanced framework for signal processing supported by physical units, that is, the plain data can be stored in a very simple number format, even fixed point numbers, but the sampling parameters rate and amplitude can be complex types, like numbers with physical units,
- frameworks for inference of sample rate and amplitude, that is, sampling rate and amplitude can be omitted in most parts of a signal processing expression. They are inferred automatically, just as types are inferred in Haskell's type system. Although the inference of signal parameters needs some preprocessing, the frameworks preserve the functional style of programming and do not need Arrows and according notation.

  We have checked three approaches, where the last one is the most promising.

  – Explicitly maintain a dictionary of signal parameters in a Reader-Writer-State monad, which must be computed completely before any signal processing takes place. This forces all signal parameters to share the same type and prohibits infinitely many signal processors to be involved (e.g., concatenation of infinitely many short noises).
  – Simulation of logic programming by lazy cycles of function applications (i.e., tied knots, fixed points). The main problems are quadratical computation complexity and a cumbersome and error-prone application. Namely, for each input you have to handle a parameter output, and vice versa for propagation of parameters through the network. You need combinators (infix operators) for combining these functions, but you will easily run into cases where you must plug manually, which is a nightmare.
  – Unify only the sample rate. Use a Reader functor/monad. Amplitude is propagated from inputs to outputs only. This is a bit conservative, but is simple and comprehensive and fulfils our needs so far.

- We checked several low-level implementations in order to achieve reasonable speed. The standard list data structure is very convenient for programming but much too slow for signal processing. We try to get rid of it in several ways:

  – A fusion framework based on `mapAccumL` and `unfoldr` like functions. Since in current GHC versions the optimisation rules do not fire reliably (e.g., rules are not specialised if a function gets specialised to a monomorphic type) we end up with intermediate list structures too often.
  – A chunky list based on the StorableVector is much faster if higher order functions like `map` and `unfoldr` are inlined. However, this data structure is not elementwise lazy (a problem for feedback), and can store only values of `Storable` type (e.g., functions are excluded).
  – A data structure analogous to the `Stream` framework, where a list is represented by a `StateT s Maybe a` which generates signal values by calling the generator function. In this approach fusion happens by inlining, and lists or other data structures can be used for sharing and feedback including sharing.

  A combination of the last two approaches seems to be a good choice so far. However, maintaining all code versions for comparison purposes led to much code duplication in the meantime.

The library comes with basic Cabal support and requires the Numeric Prelude framework (→ 5.5.4) of revised numeric type classes.

Future plans

- Design a common API to the Haskell synthesiser code, CSound support included in Haskore (→ 5.9.3), and the SuperCollider interface.
- Connect with the HaskellDSP library http://haskelldsp.sourceforge.net/. (As a beginning we have prepared and uploaded it to Hackage.)
- Hope on faster code generated by Haskell compilers. `:-)` In simple setups GHC generated code can compete with C, but in more complex ones the performance is not satisfying.

Further reading

- http://darcs.haskell.org/synthesizer/
- http://dafx04.na.infn.it/WebProc/Proc/P_201.pdf

6.4.2 hmp3

Report by:	Don Stewart
Status:	stable, maintained

hmp3 is a curses-based mp3 player frontend to mpg321 and mpg123. It is written in Haskell. It is designed to be simple, fast, and robust. It is very stable. hmp3 has been updated to version 1.5.1, and is available from Hackage.

Further reading

- Documentation can be found at: http://www.cse.unsw.edu.au/~dons/hmp3.html
- The source repository is available: `darcs get` http://www.cse.unsw.edu.au/~dons/code/hmp3/

6.4.3 Glome

Report by:	Jim Snow
Status:	experimental

Glome is a rendering engine for 3-D graphics, based on ray tracing. It was originally written in OCaml, but has since been ported (except for a few features) to Haskell, and most future development is likely to happen in Haskell.

It supports shadows and reflections, and base primitives include triangles, disks, boxes, cylinders, cones, spheres, boxes, and planes. More complex primitives can be made by grouping primitives, taking the boolean difference or intersection of primitives, or by making transformed instances.

Input and output capabilities are limited. Input is accepted as NFF-format files, or scenes may also be hard-coded in Haskell. Output is via an OpenGL window.

Rendering speed is reasonably fast, but a little too slow for interactive graphics. Glome uses a Bounding Interval Hierarchy internally to reduce the number of ray-intersection tests, so that, in general, rendering time increases logarithmically with scene complexity rather than linearly.

Further reading

http://syn.cs.pdx.edu/~jsnow/glome/

6.4.4 easyVision

Report by:	Alberto Ruiz
Status:	experimental, active development

The *easyVision* project is a collection of libraries for fast prototyping of simple computer vision and image processing applications. We take advantage of Haskell's expressive power without any performance loss, since most of the computationally expensive computations are done by optimised libraries: HOpenGL, hmatrix (→ 5.2.5), Intel's IPP and MPlayer.

The interface to most image processing functions is now purely functional, based on a first approach to automatic generation of IPP wrappers. Other recent developments include new experimental modules for Kalman filtering and geometric algebra, and more illustrative examples.

Further reading

http://alberrto.googlepages.com/easyvision

6.5 Proof Assistants and Reasoning

6.5.1 Galculator

Report by:	Paulo Silva
Status:	unstable, work in progress

The *Galculator* is a prototype of a proof assistant based on the algebra of Galois connections. When combined with the pointfree transform and tactics such as the indirect equality principle, Galois connections offer a very powerful, generic device to tackle the complexity of proofs.

The prototype of *Galculator* is being developed under an ongoing PhD project. It is still experimental and things tend to change quickly, but we are close to release the first usable version.

The source code is available from a public `SVN` repository accessible from the project homepage. After reaching its first version it will also be available from Hackage.

Currently, we are working on the automatic derivation of the so-called "free-theorems" of polymorphic functions (→ 3.3.2) and their application to proofs. Moreover, more complex constructions of Galois connections are also being studied. Finally, we plan to integrate the *Galculator* with a theorem prover, namely Coq.

Further reading

http://www.di.uminho.pt/research/galculator

6.5.2 funsat: DPLL-style Satisfiability Solver

Report by:	Denis Bueno
Status:	First release imminent, repository available

funsat (mnemonic: functional SAT solver) is a modern satisfiability solver in Haskell, intended to be competitive with state-of-the-art solvers (which are mostly written in C/C++). The strategy is to draw on many ideas from the literature and implement them in a way that is functional, testable, and difficult to accomplish

concisely in a lower-level language. Currently the emphasis is on techniques for solving structured, rather than randomised, instances.

Funsat can solve many structured instances from satlib (http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html) including PARITY (16 series), BF, blocksworld, and logistics. Many are solved in a few seconds.

The code in its current state is available as a git repository:

```
$ git clone http://churn.ath.cx/funsat
```

The immediate priority is an initial release, which should happen shortly after the end of May.

6.5.3 sat-micro-hs: SAT-Micro in Haskell

Report by:	Denis Bueno
Status:	Version 0.1.1

Sat-micro-hs is a Haskell port of the OCaml satisfiability (SAT) solver described in "SAT-Micro: petit mais costaud!" ("SAT-Micro: small but strong!", see below). The paper describes a minimal solver with the *flavour* of a modern SAT solver, without the *robustness* necessary for solving hard SAT instances. This port is intended for those interested in SAT generally, as well as any interested in the paper specifically, but who do not read French.

The code is available from Hackage at http://hackage.haskell.org/cgi-bin/hackage-scripts/package/sat-micro-hs.

Further reading

Sylvain Conchon, Johannes Kanig, and Stéphane Lescuyer. "SAT-Micro: petit mais costaud!" In *Dix-neuvièmes Journées Francophones des Langages Applicatifs*, Étretat, France, 2008. INRIA. Available online at http://www.lri.fr/~conchon/publis/conchon-jfla08.ps.

6.5.4 Saoithín: a 2nd-order proof assistant

Report by:	Andrew Butterfield
Status:	ongoing

Saoithín (pronounced "Swee-heen") is a GUI-based 2nd-order predicate logic proof assistant. The motivation for its development is the author's need for support in doing proofs within the so-called "Unifying Theories of Programming" paradigm (UTP). This requires support for 2nd-order logic, equational reasoning, and meets a desire to avoid re-encoding the theorems into some different logical form. It also provides proof transcripts whose style makes it easier to check their correctness.

Saothín is implemented in GHC 6.4 and wxHaskell 0.9.4, and has been tested on a range of Windows platforms (98/XP/Vista), and should work in principle on Linux/Mac OS X. A first public release of the software in some form is anticipated in early 2008.

Further reading

https://www.cs.tcd.ie/Andrew.Butterfield/Saoithin

6.5.5 Term Rewriting Tools written in Haskell

Report by:	Salvador Lucas

During the last years, we have developed a number of tools for implementing different termination analyses and making declarative debugging techniques available for Term Rewriting Systems. We have also implemented a small subset of the Maude/OBJ languages with special emphasis on the use of simple programmable strategies for controlling program execution and new commands enabling powerful execution modes.

The tools have been developed at the Technical University of Valencia (UPV) as part of a number of research projects. The following people are (or have been) involved in the development of these tools: Beatriz Alarcón, María Alpuente, Demis Ballis (Università di Udine), Santiago Escobar, Moreno Falaschi (Università di Siena), Javier García-Vivó, Raúl Gutiérrez, José Iborra, Salvador Lucas, Rafael Navarro, Eloy Romero, Pascal Sotin (Université du Rennes).

Status

The previous work led to the following tools:

- MU-TERM: a tool for proving termination of rewriting with replacement restrictions (first version launched in February 2002).

  http://zenon.dsic.upv.es/muterm

  Standalone versions of the tool are available for different platforms (Linux, Mac OS X, Windows). A web-based interface (developed in HAppS) is also available:

  http://zenon.dsic.upv.es/webmuterm

- Debussy: a declarative debugger for OBJ-like languages (first version launched in December 2002).

  http://www.dsic.upv.es/users/elp/debussy

- OnDemandOBJ: A Laboratory for Strategy Annotations (first version launched in January 2003).

  http://www.dsic.upv.es/users/elp/ondemandOBJ

  http://www.dsic.upv.es/users/elp/GVerdi

- GVerdi: A Rule-based System for Web site Verification (first version launched in January 2005).

All these tools have been written in Haskell (mainly developed using Hugs and GHC) and use popular Haskell libraries like HAppS, hxml-0.2, Parsec, RegexpLib98, wxHaskell.

Immediate plans

Improve the existing tools in a number of different ways and investigate mechanisms (XML, .NET, ...) to plug them to other client/server applications (e.g., compilers or complementary tools).

References

- Building .NET GUIs for Haskell applications. B. Alarcón and S. Lucas. 6th International Conference on .NET Technologies, pages 57–66, 2006.
- Proving Termination of Context-Sensitive Rewriting With MU-TERM B. Alarcón, R. Gutiérrez, J. Iborra, and S. Lucas. Electronic Notes in Theoretical Computer Science, 118:105-115, 2007.
- Abstract Diagnosis of Functional Programs M. Alpuente, M. Comini, S. Escobar, M. Falaschi, and S. Lucas Selected papers of the International Workshop on Logic Based Program Development and Transformation, LOPSTR'02, LNCS 2664:1–16, Springer-Verlag, Berlin, 2003.
- OnDemandOBJ: A Laboratory for Strategy Annotations M. Alpuente, S. Escobar, and S. Lucas 4th International Workshop on Rule-based Programming, RULE'03, Electronic Notes in Theoretical Computer Science, volume 86.2, Elsevier, 2003.
- Connecting Remote Tools: Do it by yourSELF! M. Alpuente and S. Lucas. ERCIM News 61:48–49, April 2005.
- MU-TERM: A Tool for Proving Termination of Context-Sensitive Rewriting S. Lucas 15th International Conference on Rewriting Techniques and Applications, RTA'04, LNCS 3091:200–209, Springer-Verlag, Berlin, 2004.
- A Rule-based System for Web site Verification. Demis Ballis and Javier García-Vivó. 1st International Workshop on Automated Specification and Verification of Web Sites, WWV'05, Valencia (SPAIN). Electronic Notes in Theoretical Computer Science, 157(2):11–17, 2006.

6.5.6 Inference Services for Hybrid Logics

Report by:	Carlos Areces
Participants:	Daniel Gorin, Guillaume Hoffmann

"Hybrid Logic" is a loose term covering a number of logical systems living somewhere between modal and classical logic. For more information on these languages, see http://hylo.loria.fr

The Talaris group at Loria, Nancy, France (http://talaris.loria.fr) and the GLyC group at the Computer Science Department of the University of Buenos Aires, Argentina (http://www.glyc.dc.uba.ar/) are developing a suite of tools for automated reasoning for hybrid logics, available at http://hylo.loria.fr/intohylo/. Most of them are (successfully) written in Haskell. See HyLoRes (→ 6.5.7), HTab (→ 6.5.8), and HGen (→ 6.5.9).

6.5.7 HyLoRes

Report by:	Carlos Areces
Participants:	Daniel Gorin, Guillaume Hoffmann
Status:	active development
Current release:	2.4

HyLoRes is an automated theorem prover for hybrid logics (→ 6.5.6) based on a resolution calculus. It is sound and complete for a very expressive (but undecidable) hybrid logic, and it implements termination strategies for certain important decidable fragments. The project started in 2002, and has been evolving since then. It is currently being extended to handle even more expressive logics (including, in particular, temporal logics). In the near future, we will investigate algorithms for model generation.

The source code is available. It is distributed under the terms of the Gnu GPL.

Further reading

- Areces, C. and Gorin, D. *Ordered Resolution with Selection for H(@).* In Proceedings of LPAR 2004, pp. 125–141, Springer, Montevideo, Uruguay, 2005.
- Areces, C. and Heguiabehere, J. *HyLoRes: A Hybrid Logic Prover Based on Direct Resolution.* In Proceedings of Advances in Modal Logic 2002, Toulouse, France, 2002.
- Site and source: http://hylo.loria.fr/intohylo/hylores.php

6.5.8 HTab

Report by:	Carlos Areces
Participants:	Daniel Gorin, Guillaume Hoffmann
Status:	active development
Current release:	1.2.2

HTab is an automated theorem prover for hybrid logics (→ 6.5.6) based on a tableau calculus. The goal is to implement a terminating tableau algorithm for the basic hybrid logic and for the basic logic extended with the universal modality. It is currently in early development. It will be tunable with various optimisations.

The source code is available. It is distributed under the terms of the Gnu GPL.

Further reading

- Hoffmann, G. and Areces, C. *HTab: a terminating tableaux system for hybrid logic.* In Methods for Modalities 5, Cachan, France, 2007.
- Site and source: http://hylo.loria.fr/intohylo/htab.php

6.5.9 HGen

Report by:	Carlos Areces
Participants:	Daniel Gorin, Guillaume Hoffmann
Status:	active development
Current release:	1.1

HGen is a random CNF (conjunctive normal form) generator of formulas for different hybrid logics. It is highly parametrised to obtain tests of different complexity for the different languages. It has been extensively used in the development of HyLoRes (→ 6.5.7) and HTab (→ 6.5.8).

The source code is available. It is distributed under the terms of the Gnu GPL.

Further reading

- Areces, C. and Heguiabehere, J. *hGen: A Random CNF Formula Generator for Hybrid Languages*. In Methods for Modalities 3 (M4M-3), Nancy, France, September 2003.
- Site and source: http://hylo.loria.fr/intohylo/hgen.php

6.6 Modelling and Analysis

6.6.1 Coconut

Report by:	Wolfram Kahl
Participants:	Christopher K. Anand
Status:	on-going development

Coconut (COde CONstructing User Tool) is a special-purpose compiler project aiming to provide a coherent tool bench for the development of high-performance, high-assurance scientific computation, and to cover the full range of development activity from mathematical modelling to verification.

Development of the integrated tool bench is in (GHC-)Haskell, and currently proceeding from the bottom up. As code generation target, we are for now focusing on support for the Cell BE, and in particular the special-purpose SPU compute engines of which there are eight on a single Cell BE chip.

A type-indexed embedded DSL for declarative assembly language includes complex SIMD-isation patterns which are easier to encapsulate in the DSL and apply across 30 functions than they would be to implement even for a single function in C. To support rapid prototyping, Coconut includes instruction semantics sufficient to simulate SIMD instruction execution within Haskell. This significantly reduces the time spent on developing new patterns and exploring edge cases for mixed fixed/floating point computations by debugging and unit testing right in GHCi.

The central internal representation are "code graphs", which are used to represent both data-flow graphs and control flow graphs, with separate levels of nesting for non-concurrent and concurrent control flow. For scheduling simple loop bodies programmed in the DSL, we use our Explicitly Staged Software Pipelining (ExSSP) algorithm on the data-flow code graph representation.

Our implementation of single-precision special functions (sin, sinh, asin, ..., sqrt, cbrt, exp, log, lgamma, ...) for the SPU is distributed as MASS in the Cell BE SDK 3.0 in both generated C (for inlining) and long vector functions scheduled by Coconut. In comparison with a state-of-the-art hand-tuned C implementation of these library functions using in-line assembly in the form of intrinsic functions and scheduled by the compiler `spu-xlc`, the Coconut-generated and -scheduled implementations are roughly four times faster; in many cases we know that our implementations are optimal.

We are currently making good progress on novel control flow patterns and scheduling algorithms to support them, a virtual machine model for multicore architectures, and verification strategies for SIMD-isation and multicore synchronisation, in an effort to bring the level of usability we have achieved with the DSL for SIMD-isation also to multicore parallelism.

Further reading

http://coconut.mcmaster.ca/

6.6.2 Streaming Component Combinators

Report by:	Blažević Mario
Status:	experimental, actively developed

Streaming Component Combinators are an experiment at modelling dataflow architecture by using composable streaming components. All components are categorised into a small set of component types. A number of components can be composed into a compound component using a component combinator. For example, two transducer components can be composed together using a *pipe* operator into another transducer; one splitter and two transducers can be composed using an *if* combinator into a single compound transducer. Components are implemented as coroutines, and the data flow among them is synchronous.

There are two ways to use SCC: as an embedded language in Haskell, or as a set of commands in a command-line shell. The latter provides its own parser and a rudimentary type checker, but otherwise relies on the former to do the real work.

The original work was done in the OmniMark programming language. Haskell was the language of choice for the second implementation because its strong typing automatically makes the embedded language strongly typed, and because its purity forces the implementation to expose the underlying semantics.

The currently planned future work includes extending the set of primitive components and component

combinators, and improving the type-checking and scripting abilities of the shell interface.

The latest stable version of SCC is available from Hackage.

Further reading

- Hackage: http://hackage.haskell.org/cgi-bin/hackage-scripts/package/scc-0.1
- Conference paper: Mario Blažević, Streaming component combinators, Extreme Markup Languages, 2006. http://www.idealliance.org/papers/extreme/proceedings/html/2006/Blazevic01/EML2006Blazevic01.html
- OmniMark implementation: http://developers.omnimark.com/etcetera/streaming-component-combinators.tar.gz

6.6.3 Raskell

Report by:	Jennifer Streb
Participants:	Garrin Kimmell, Nicolas Frisby, Mark Snyder, Philip Weaver, Perry Alexander
Status:	beta, actively maintained

Raskell is a Haskell-based analysis and interpretation environment for specifications written using the system-level design language Rosetta. The goal of Rosetta is to compose heterogeneous specifications into a single semantic environment. Rosetta provides modelling support for different design domains employing semantics and syntax appropriate for each. Therefore, individual specifications are written using semantics and vocabulary appropriate for their domains. Information is then composed across these domains by defining interactions between them.

The heart of Raskell is a collection of composable interpreters that support type checking, evaluation, and abstract interpretation of Rosetta specifications. Algebra combinators allow semantic algebras for the same constructs, but for different semantics, to be easily combined. This facilitates further reuse of semantic definitions. Comonads are used to structure a denotation of temporal Rosetta specifications. We are also investigating the use of comonads to capture other models of computation as supported by Rosetta domains. Using abstract interpretation, we can transform specifications between semantic domains without sacrificing soundness. This allows for analysis of interactions between two specifications written in different semantic domains. Raskell also includes a Parsec-based Rosetta parser that generates both recursive and non-recursive AST structures.

The Raskell environment is available for download at the links below. It is continually being updated, so we recommend checking back frequently for updates. To build the Rosetta parser and type checker, you must also install InterpreterLib (→ 5.4.7) and algc (a preprocessor for functorial boilerplate), both available at the third link listed below.

Further reading

- http://www.ittc.ku.edu/Projects/SLDG/projects/project-rosetta.htm#raskell
- http://www.ittc.ku.edu/Projects/SLDG/projects/project-raskell.htm
- http://www.ittc.ku.edu/Projects/SLDG/projects/project-InterpreterLib.htm

Contact

⟨alex@ittc.ku.edu⟩

6.6.4 VooDooM

Report by:	Tiago Miguel Laureano Alves
Participants:	Paulo Silva
Status:	stable, maintained

VooDooM supports understanding and re-engineering of VDM-SL specifications.

Understanding is accomplished through the extraction and derivation of different kinds of graphs such as type dependency, function dependency, and strongly connected components graphs. These graphs can be subject of both visualisation (by exporting into DOT format) and metrication (generating CSV or text report).

Re-engineering is supported through the application of transformation rules to the datatypes to obtain an equivalent relational representation. The relational representation can be exported as VDM-SL datatypes (inserted back into the original specification) and/or SQL table definitions (can be fed to a relational DBMS).

The first VooDooM prototype, supporting re-engineering, was developed in a student project by Tiago Alves and Paulo Silva. The prototype was further enhanced and continued as an open source project (http://voodoom.sourceforge.net/) in the context of the IKF-P project (Information Knowledge Fusion, http://ikf.sidereus.pt/) by Tiago Alves and finally in the context of a MSc thesis project.

Future plans

It is planned that the re-engineering functionality of VooDooM will be replaced by the one that is being developed for the 2LT project (→ 5.6.5), which will add XML and Haskell generation.

Further reading

VooDooM is available from http://voodoom.sourceforge.net/. The implementation of VooDooM makes ample use of strategic programming, using Strafunski, and is described in *Strategic Term Rewriting*

and Its Application to a VDM-SL to SQL Conversion (Alves et al., Formal Methods 2005) and in the MSc thesis *VooDooM: Support for understanding and re-engineering of VDM-SL specifications.*

6.7 Specialised Domains

6.7.1 A Survey on the Use of Haskell in Natural-Language Processing

Report by: Richard A. Frost

The survey "Realization of Natural-Language Interfaces Using Lazy Functional Programming" was published in ACM Computing Surveys, Volume 38, Issue 4, Article 11, in December 2006. It was in the Top 10 downloads from ACM Magazines and Surveys for February and March of 2007. The survey is now being used at a few universities as required reading for courses on computational linguistics.

The survey currently contains 168 references to relevant publications. The survey will be updated as more information becomes available. Please send information on recent publications in this area to ⟨rfrost@cogeco.ca⟩

Further reading

A draft of the survey is available at:
http://cs.uwindsor.ca/~richard/PUBLICATIONS/NLI_LFP_SURVEY_DRAFT.pdf

6.7.2 GenI

Report by: Eric Kow

GenI is a surface realiser for Tree Adjoining Grammars. Surface realisation can be seen as the last stage in a natural language generation pipeline. GenI in particular takes an FB-LTAG grammar and an input semantics (a conjunction of first order terms), and produces the set of sentences associated to the input semantics by the grammar. It features a surface realisation library, several optimisations, 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.

GenI has recently been updated to compile on GHC 6.8.2. It is also available on Hackage, and can be installed via cabal-install. We also now have a mailing list at http://websympa.loria.fr/wwsympa/info/geni-users.

Further reading

- http://trac.loria.fr/~geni
- Paper from Haskell Workshop 2006: http://hal.inria.fr/inria-00088787/en

6.7.3 Bioinformatics tools

Report by: Ketil Malde

The Haskell bioinformatics library supports working with nucleotide and protein sequences and associated data. File format support includes sequences in Fasta (with associated quality information), TwoBit, and PHD formats, BLAST XML output, and ACE alignment files.

The standard alignment algorithms (and some non-standard ones) are provided, as well as sequence indexing, complexity calculation, protein translation, etc.

The library is considered in development (meaning things will be added, some functionality may not be as complete or well documented as one would wish, and so on), but central parts should be fairly well documented and come with a QuickCheck test and benchmarking suite.

The library abstracts functionality that is used in a handful of applications, including:

- `xsact` — an EST clustering program
- `RBR` — a repeat detector/masker
- `clusc` — a tool for calculating cluster similarity with a bunch of metrics
- `dephd` — a sequence quality assessment tool
- `xml2x` — a BLAST postprocessor and GO annotator

Everything is GPLed and available as Darcs repos, at http://malde.org/~ketil/biohaskell/.

6.7.4 Inductive Programming

Report by: Lloyd Allison

Inductive Programming (IP): The learning of general hypotheses from given data.

The project is (i) to use Haskell to examine what are the products of artificial-intelligence (AI)/data mining/machine-learning from a programming point of view, and (ii) to do data analysis with them.

IP 1.2 now contains estimators, from given weighted and unweighted data, to the Poisson and Geometric distributions over non-negative integer variables, and Student's t-Distribution over continuous variables. The new (and the earlier) distributions may be used as components to the learners (estimators) of structured models such as unsupervised classifications (mixture models), classification- (decision-, regression-) trees and other function-models (regressions), mixed Bayesian networks, and segmentation models. IP's modules have also been slightly reorganised.

A small prototype module of numerical/scientific functions, in Haskell, has been added to IP 1.2, to support the implementation of Student's t-Distribution in the first instance.

Prototype code is available (GPL) at the URL below.

Future plans

Planned are continuing extensions, applications to real data-sets, and comparisons against other learners.

Further reading

- http://www.allisons.org/ll/FP/IP/
- http://www.csse.monash.edu.au/~lloyd/tildeFP/II/

6.8 Others

6.8.1 lambdabot

Report by:	Don Stewart
Status:	active development

lambdabot is an IRC robot with a plugin architecture, and persistent state support. Plugins include a Haskell evaluator, lambda calculus interpreter, unlambda interpreter, pointfree programming, dictd client, fortune cookies, Google search, online help, and more.

lambdabot 4.0 has been released, and is available from Hackage. Cale Gibbard has also kindly taken over maintainance of the bot.

Further reading

- Documentation can be found at: http://www.cse.unsw.edu.au/~dons/lambdabot.html
- The source repository is available: `darcs get` http://code.haskell.org/lambdabot

6.8.2 FreeArc

Report by:	Bulat Ziganshin
Status:	beta

At this moment, FreeArc is the best practical archiver in the world, providing the maximum speed/compression ratio (http://www.maximumcompression.com/data/summary_mf2.php).

Besides this, FreeArc provides a lot of features, including solid archives with fast updates, tunable compression algorithms, support for external compressors, automatic selection of compression algorithm depending on file type, data encryption and recovery, Win32 and Linux versions, tunable sorting and grouping of files, and FAR/Total Commander MultiArc support.

Such an ambitious goal was accomplished by bringing together Haskell and C++: speed-critical parts (compression, encryption) are written in C++, while everything else benefits from fast development and high reliability opportunities provided by Haskell. I should also note that compared to other archivers (traditionally written in C++) FreeArc provides smarter algorithms of archive management, which is again due to the high level of the Haskell programming paradigm.

Program sources are open, so you can borrow there:

- compression libraries which include 11 compression algorithms with easy Haskell interface (http://haskell.org/haskellwiki/Library/Compression)
- encryption code which provides AES, Blowfish, Twofish, and Serpent encryption algorithms with all the bells and whistles (PRNG, PBKDF, SHA512) required for really secure encryption of data streams

Moreover, the program includes a few more modules that you may reuse in your program on BSD3 license:

- Win32Files.hs — implements I/O on Windows for files > 4GB and files with Unicode names
- Files.hs — provides an OS-independent interface to the features of Win32Files
- Charsets.hs — encode/decode data in OEM, ANSI, UTF-8/16/32 encodings
- ByteStream.hs — binary serialisation library
- UTF8Z.hs — UTF8-packed strings (like ByteString, but with a more memory-efficient representation)
- Process.hs — allows to construct data-processing algorithms from individual processes by joining them together very much like ordinary programs are joined by Unix shell

The program is extensively commented in Russian, so for Russian-speaking Haskellers it may be an invaluable source for learning "practical Haskell".

Further reading

Download: http://freearc.sourceforge.net

Contact

⟨Bulat.Ziganshin@gmail.com⟩

6.8.3 Roguestar

Report by:	Christopher Lane Hinson
Status:	early development

Roguestar is a science fiction themed roguelike (turn-based, chessboard-tiled, role playing) game written in Haskell. Roguestar uses OpenGL for graphics.

Roguestar 0.2 was announced on May 9 2008, and featured a simple combat system.

RSAGL, the RogueStar Animation and Graphics Library, includes a domain-specific monad for 3D modelling of arbitrary parametric surfaces, as well as an animation monad and arrow, which is more or less like YAMPA as a stack of arrow transformers. RSAGL was specifically designed for roguestar, but every effort has been made (including the license) to make it accessible to other projects that might benefit from it.

Roguestar is licensed under the Affero General Public License. RSAGL is licensed under a permissive license.

Further reading

http://roguestar.downstairspeople.org

7 Commercial Users

7.1 Well-Typed LLP

Report by: Ian Lynagh
Participants: Björn Bringert, Duncan Coutts

Well-Typed is a company which provides consulting services to users of Haskell. The company was incorporated in March 2008, with Björn Bringert, Duncan Coutts, and Ian Lynagh making up the founding partners.

We offer a broad range of services, from initial project design and training to development, maintenance, and performance tuning, and everything in between.

For more information, please take a look at our website, or drop us an e-mail at ⟨info@well-typed.com⟩.

Further reading

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

7.2 SeeReason Partners, LLC

Report by: Clifford Beshers
Participants: David Fox, Jeremy Shaw

Clifford Beshers, David Fox, and Jeremy Shaw have formed SeeReason Partners, LLC. Our plan is to deliver services over the internet, using Haskell to build our applications whenever possible. We are looking at a variety of niche domains, including teaching primary mathematics skills and art appraisal, seeking to create sites that support social networking, learning, and digital asset management.

Often such projects employ large teams of developers and artists to hand-craft a look and feel. We want to exploit functional programming to generate content, or at least distribute the labour, more efficiently. We are exploring both AJAX technologies and Macromedia Flash. We are working on a Haskell to Flash compiler, as well as libraries to construct applications with Adobe's Flex UI toolkit.

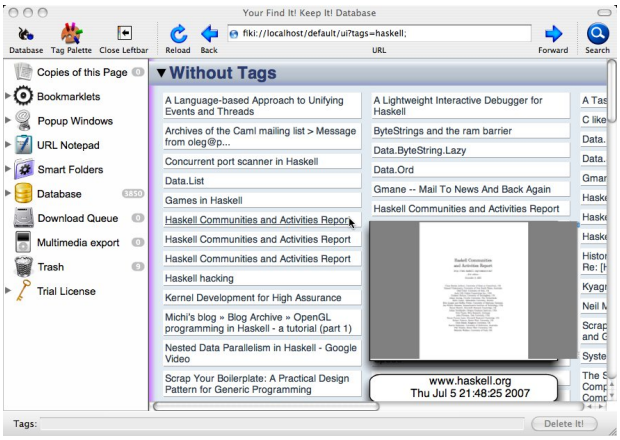
Formerly core members of the operating systems group at Linspire, Inc., we continue to maintain the tools for managing a Debian Linux distribution that we developed there. Source code for these tools can be found at our public source code repository http://src.seereason.org/. We plan to use these tools to provide current archives of Haskell related packages.

We can be reached at ⟨(cliff,david,jeremy)@seereason.com⟩ and on `#haskell` (→ 1.2) respectively as thetallguy, dsfox, and stepcut.

7.3 Ansemond LLC

Report by: Sengan Baring-Gould

Find It! Keep It! is a Mac Web Browser that lets you keep the pages you visit in a database. A list of these pages is shown in the "database view". This view is rendered by the browser from generated HTML and is dynamically updated by Javascript DOM operations: tens of thousands of elements cannot efficiently be placed on the screen using DOM operations only, while rerendering a half a megabyte of HTML each time a user interface element changes is unresponsive. A glitch free user experience requires keeping these two separate mechanisms synchronised, which proved difficult.



A new Haskell implementation generates abstract DOM operations which are then either rendered to HTML or are converted to Javascript DOM operations to be run within the browser. While this process is not complex (difficult algorithm) it is complicated (hard for the programmer to keep everything in mind). The additional modularity afforded by laziness proved invaluable, enabling all the different pieces to be coded much more independently and clearly than was possible in the original Python version. This same engine could be used on a web server and would work with any web browser. The Haskell version is scheduled to ship in version 1.1 of *Find It! Keep It!*

Ansemond LLC is at http://www.ansemond.com.

7.4 Credit Suisse Global Modelling and Analytics Group

Report by: Ganesh Sittampalam

GMAG, the quantitative modelling group at Credit Suisse, has been using Haskell for various projects since

the beginning of 2006, with the twin aims of improving the productivity of modellers and making it easier for other people within the bank to use GMAG models.

Many of GMAG's models use Excel combined with C++ addin code to carry out complex numerical computations and to manipulate data structures. This combination allows modellers to combine the flexibility of Excel with the performance of compiled code, but there are significant drawbacks: Excel does not support higher-order functions and has a rather limited and idiosyncratic type system. It is also extremely difficult to make reusable components out of spreadsheets or subject them to meaningful version control.

Because Excel is (in the main) a side-effect free environment, functional programming is in many ways a natural fit, and we have been using Haskell in various ways to replace or augment the spreadsheet environment.

Our past projects include:

- Adding higher-order functions to Excel, implemented via (Haskell) addin code.
- Tools to transform spreadsheets into directly executable code.
- A "lint" tool to check for common errors in spreadsheets.

Our main current project is Paradise, a domain-specific language embedded in Haskell for implementing reusable components that can be compiled into multiple target forms. It has been under development for over 18 months now, and over that time the team working on it has grown to several people.

Paradise's first target form was Excel spreadsheets, and that backend is now relatively mature; our main focus at the moment is generating .NET components which can be run standalone or plugged into a bank-wide system. In future we may target yet more diverse platforms such as web browsers.

Several modellers have now been exposed directly to Haskell by using Paradise, and they have generally picked it up fairly quickly. All new recruits are now introduced to Haskell as part of our internal training program.

Further reading

CUFP 2006 talk about Credit Suisse:
http://cufp.galois.com/slides/2006/HowardMansell.pdf

7.5 Barclays Capital Quantitative Analytics Group

Report by:	Simon Frankau

Barclays Capital's Quantitative Analytics group is using Haskell to develop an embedded domain-specific functional language (called "FPF") which is used to specify exotic equity derivatives. These derivatives, which are naturally best described in terms of mathematical functions, and constructed compositionally, map well to being expressed in an embedded functional language. This language is now regularly being used by people who had no previous functional language experience.

The FPF description then serves as the core description of the trade structure, with different interpretations being used to perform operations from generating pricing instructions for the bank's risk systems through to generating trade input forms and reports. The system thus automates the introduction of new products, replacing the previous approach of manually extending each subsystem to cope with a new trade type manually. It has dramatically reduced the turnaround time to make a new trade production-ready.

We have found Haskell to be a great language in which to implement an embedded functional language, and to be very effective as a language for rapid development.

We have been working on the system for a little over a year, and it has been in production use for most of that time, with new features and interpretations being added incrementally over that time.

We are hiring. Please contact Simon Frankau ⟨Simon.Frankau@barcap.com⟩ for more details.

7.6 Bluespec tools for design of complex chips

Report by:	Rishiyur Nikhil
Status:	Commercial product

Bluespec, Inc. provides tools for chip design, modelling, and verification (ASICs and FPGAs), inspired by Haskell and Term Rewriting Systems. Bluespec also uses Haskell to implement many of its tools (over 100K lines of Haskell). Bluespec's products include synthesis, simulation, and other tools for Bluespec SystemVerilog (BSV).

Bluespec's customers are using BSV's high expressivity and full synthesisability to move into hardware many functions previously thought to be feasible only in software (complex algorithms, testbenches, and models).

BSV is based on the following semantic model: hardware behaviour is expressed using *Rewrite Rules*, and inter-module communication is expressed using *Rule-based Interface Methods* (which allow rules to be composed from fragments that span module boundaries). Because rules are atomic transactions, they eliminate a majority of the "timing errors" and "race conditions" that plague current hardware design using existing RTL languages like Verilog or VHDL. Rules also

enable powerful reasoning about the functional correctness of systems. In other words, the concurrency model provided by rules is much more powerful and abstract than the low-level concurrency models provided by Verilog, VHDL, and SystemC.

BSV incorporates Haskell-style polymorphism and overloading (typeclasses) into SystemVerilog's type system. BSV also treats modules, interfaces, rules, functions, etc. as first-class objects, permitting very powerful static elaboration (including recursion).

Bluespec tools synthesise source code into clocked synchronous hardware descriptions (in Verilog RTL) that can be simulated or further synthesised to netlists using industry-standard tools. This automates the generation of control logic to manage complex concurrent state update, a major source of errors in current design methodology, where this logic must be manually coded by the designer.

The powerful Haskell-like static elaboration in BSV is *control adaptive*, i.e., a parameterised design can elaborate into different microarchitectures which have different resource conflicts, but the typically complex control logic necessary to manage these conflicts is synthesised automatically based on the atomic transaction semantics — this is key to BSV's high level of abstraction.

Bluespec participates in standards committees like IEEE P1800 (SystemVerilog) and IEEE P1666 (SystemC), where it tries to encourage adoption of the declarative programming ideas in BSV. One success has been the adoption of Bluespec's proposals for "tagged unions (algebraic types) and pattern matching" in the current IEEE SystemVerilog standard.

Status

Bluespec SystemVerilog and its tools have been available since 2004. The tools are now in use by several major semiconductor companies (see Bluespec website or contact Bluespec for details) and several universities (including MIT, CMU, UT Austin, Virginia Tech, Indian Institute of Science, and U. Tokyo).

Availability

Bluespec SystemVerilog tools are sold commercially by Bluespec, Inc., which also makes all its tools available at no charge to academic institutions for teaching and research.

Some historical notes and acknowledgements

The technology for synthesising from Term Rewriting Systems to competitive RTL was originally developed by James Hoe and Prof. Arvind at MIT in the late 1990s. At Sandburst Corp., during 2000–2003, Lennart Augustsson was the principal designer of "Bluespec Classic", the first "industrial strength" variant of the language, with Rishiyur Nikhil, Joe Stoy, Mieszko Lis, and Jacob Schwartz contributing to language and tool development and use. The latter four continued work on BSV at Bluespec, Inc. from 2003 with additional contributions from Ravi Nanavati, Ed Czeck, Don Baltus, Jeff Newbern, Elliot Mednick, and several summer interns.

Further reading

- Company website and wiki: http://www.bluespec.com, http://www.bluespec.com/wiki
  A series of small illustrative examples: http://www.bluespec.com/wiki/SmallExamples
- Winning entry in MEMOCODE 2008 contest to decrypt, sort and re-encrypt a database of records; 1100 times faster than the reference software, 11 times faster than the runner-up; the only entry to be implemented entirely in hardware (FPGA), written in BSV: http://rijndael.ece.vt.edu/memocontest08/
- Publications:
  http://www.bluespec.com/technology/research.htm
  Bringing Declarative Programming into a Commercial Tool for Developing Integrated Circuits, Rishiyur Nikhil, Commercial Users of Functional Programming (CUFP), September 2006, slides of presentation at http://www.galois.com/cufp/
  MIT courseware, "Complex Digital Systems":
  http://csg.csail.mit.edu/6.375
  A fun example with lots of functional-programming features — BluDACu, a parameterised Bluespec hardware implementation of Sudoku:
  http://www.bluespec.com/products/BluDACu.htm

7.7 Galois, Inc.

Report by: Andy Adams-Moran

Galois is an employee-owned software development company based in Beaverton, Oregon, U.S.A. Galois started in late 1999 with the stated purpose of using functional languages to solve industrial problems. These days, we emphasise the needs of our clients and their problem domains over the techniques, and the slogan of the Commercial Users of Functional Programming Workshop (see http://cufp.functionalprogramming.com/) exemplifies our approach: Functional programming as a *means*, not an *end*.

Galois develops software under contract, and every project (bar three) that we have ever done has used Haskell. The exceptions used ACL2, Poly-ML, SML-NJ, and OCaml, respectively, so functional programming languages and formal methods are clearly our "secret sauce". We deliver applications and tools to clients in industry and the U.S. government. Some diverse examples: Cryptol, a domain-specific language for cryptography (with an interpreter and a compiler, with mul-

tiple targets, including FPGAs); a GUI debugger for a specialised microprocessor; a specialised, high assurance, cross-domain web and file server, and wiki for use in secure environments, and numerous smaller research projects that focus on taking cutting-edge ideas from the programming language and formal methods community and applying them to real world problems.

Web-based technologies are increasingly important to our clients, and we believe Haskell has a key role to play in the production of reliable, secure web software. The culture of correctness Haskell encourages is ideally suited to web programming, where issues of security, authentication, privacy, and protection of resources abound. In particular, Haskell's type system makes possible strong static guarantees about access to resources, critical to building reliable web applications.

To help push further the adoption of Haskell in the domain of web programming, Galois released a suite of Haskell libraries, including:

- json: Support for JavaScript Object Notation
- xml: A simple, lightweight XML parser/generator.
- utf8-string: A UTF8 layer for IO and Strings.
- selenium: Communicate with a Selenium Remote Control server.
- curl: libcurl is a rich client-side URL transfer library.
- sqlite: Haskell binding to sqlite3 databases.
- feed: Interfacing with RSS and Atom feeds
- mime: Haskell support for working with MIME types.

Continuing our deep involvement in the Haskell community, Galois was happy to sponsor the two Haskell hackathons held in the past year, Hac 07 II, in Freiburg, Germany, and Hac4 in Gothenburg, Sweden. Galois also sponsored the second BarCamp Portland, held in early May 2008.

Further reading

http://www.galois.com/.

8 Research and User Groups

8.1 Functional Programming Lab at the University of Nottingham

Report by: Wouter Swierstra

The School of Computer Science at the University of Nottingham has recently formed the *Functional Programming Laboratory*, a new research group focused on all aspects of the theory and practice of functional programming, together with related topics such as type theory, category theory, and quantum programming. The laboratory is lead jointly by Thorsten Altenkirch and Graham Hutton, and comprises Neil Ghani and Henrik Nilsson as academic members of staff, and currently 3 research assistants and 10 PhD students. The group's weekly meetings, which started in 2004, are recorded on the frequently cited *FP lunch blog* (http://sneezy.cs.nott.ac.uk/fplunch/weblog/), giving a lively picture of ongoing research at Nottingham. With a total of 20 people in the area, we have a spectrum of interests:

Application of Category Theory

The Nottingham group is active in applying ideas from category theory to practical problems in functional programming. Mauro Jaskelioff and Neil Ghani are working on modularity for structured operational semantics. (See also the following sections on initial algebra semantics and containers.)

Containers

Nottingham is home to the EPSRC grant on *containers*, a semantic model of functional data structures. Neil Ghani, Thorsten Altenkirch, Peter Hancock, Peter Morris, and Rawle Prince are working with containers to both write and reason about programs. Peter Morris has recently finished his PhD, which used containers as a basis for generic programming with dependent types.

Datatype-Generic Design Patterns

Ondrej Rypacek together with Roland Backhouse and Henrik Nilsson are working on formal reasoning about object-oriented designs with emphasis on algebraic and datatype-generic methods. Our goal is a sound mathematical model allowing us to disclose and formalise correspondences between object-oriented and functional programming.

Dependently-Typed Haskell

Supported by a Microsoft Research studentship, Robert Reitmeier is working on integrating dependent types in Haskell under the supervision of Thorsten Altenkirch, with advice from Simon Peyton Jones. Together with Nicolas Oury we are designing an alternative dependently-typed intermediate language, influenced by our experiences with Epigram.

Epigram

Epigram (→ 3.2.3) is a dependently-typed functional programming language in its second incarnation, implemented in Haskell. With advice from Conor McBride the Epigram team Thorsten Altenkirch, James Chapman, Peter Morris, Wouter Swierstra, and Nicolas Oury are working on both practical and theoretical aspects of the language.

Functional Reactive Programming

Yampa, the latest Haskell-based implementation of Functional Reactive Programming (FRP), is currently being maintained by Henrik Nilsson. Under his supervision, Neil Sculthorpe is working on efficient scalable implementation techniques for a Yampa-like language, while George Giorgidze is applying the advantages of FRP to non-causal modelling languages. The latter approach is called Functional Hybrid Modelling (→ 5.9.2).

Functional Specifications of Effects

Wouter Swierstra and Thorsten Altenkirch have been researching pure specifications of several functions in the IO monad. This research has resulted in the Test.IOSpec library (→ 5.7.4), which may be of interest to anyone who wants to debug, reason about, analyse, or test impure code. Besides implementing these ideas in Haskell, the specifications can be made *total* in the richer type theories underlying Epigram, Coq, and Agda 2 (→ 3.2.2).

Initial Algebra Semantics

Neil Ghani has, with Patricia Johann, been working on the initial algebra semantics of advanced data types. They showed that there is no need for the generalised folds in the literature, as the standard fold from initial algebra semantics, when coupled with Right Kan extensions, is expressive enough. Interestingly, Left Kan extensions can be employed to give an initial algebra semantics for GADTs. They also used the characterisation of initial algebras as limits to give short cut fusion rules for both nested data types and GADTs.

Quantum Programming

Thorsten Altenkirch and Alex Green are working on a Haskell library to interface or simulate a hypothetical quantum computer — the Quantum IO monad. This is related and inspired by earlier work on the implementation of QML with Jonathan Grattage, who finished his PhD on the subject in 2006.

Reasoning About Programs

Supported by a grant from EPSRC, Graham Hutton, Nils Anders Danielsson, and Diana Fulger have recently started a new project on reasoning about exceptions and interrupts. This project is also closely related to Liyang HU's ongoing research on reasoning about concurrent systems using software transactional memory. During a sabbatical at Galois (→ 7.7) in the summer of 2007, Graham Hutton worked with Andy Gill on the worker/wrapper transformation.

Teaching

Haskell plays an important role in the undergraduate programme at Nottingham, as well as our China and Malaysia campuses. Modules on offer include Functional Programming, Advanced Functional Programming, Mathematics for Computer Science, Principles of Programming Languages, Compilers, and Computer-Aided Formal Verification, among others.

Programming in Haskell

Graham Hutton has written an introductory Haskell textbook (→ 1.6.1), published by Cambridge University Press, 2007.

Events

The group in Nottingham plays a leading role in the Midlands Graduate School in the Foundations of Computing Science, the British Colloquium for Theoretical Computer Science, and the Fun in the Afternoon seminar series on functional programming.

FP Lunch

Every Friday, Nottingham's functional programmers gather for lunch with helpings of informal, impromptu-style whiteboard talks. Lecturers, PhD students, and visitors are invited to discuss recent developments, problems, or projects of relevance. We blog summaries of recent talks.

In the afternoon there is an hour-long seminar. We are always keen on speakers in any related areas: do get in touch with Neil Ghani ⟨nxg@cs.nott.ac.uk⟩ if you would like to visit our group. See you there!

Further reading

- Functional Programming at Nottingham: http://fop.cs.nott.ac.uk/fp/
- Epigram: http://e-pig.org/
- Quantum Programming: http://fop.cs.nott.ac.uk/qml/
- Yampa: http://haskell.org/yampa/
- Fun in the Afternoon: http://fop.cs.nott.ac.uk/fun/
- Midlands Graduate School: http://cs.nott.ac.uk/MGS/
- FP Lunch: http://fop.cs.nott.ac.uk/fplunch/

8.2 Artificial Intelligence and Software Technology at JWG-University Frankfurt

Report by:	David Sabel
Participants:	Manfred Schmidt-Schauß

Equivalence of Call-by-Name and Call-by-Need

Haskell has a call-by-name semantics, but all efficient implementations of Haskell use call-by-need evaluation, avoiding multiple evaluations of the same expression. We showed equivalence of call-by-name and call-by-need for a tiny deterministic `letrec`-calculus and also the correctness of an unrestricted copy-reduction in both calculi. We also proved that our method scales up to extended `letrec`-calculi with `case` and constructors, as well as `letrec`-calculi with a parallel-or operator.

Semantics for Haskell extended with direct-call I/O

We introduced the calculus *FUNDIO*, which proposes a non-standard way to combine lazy functional languages with Input/Output using non-deterministic constructs. Program equivalence is based on the operational semantics including the input/output behaviour of reduction sequences. We proved correctness of a considerable set of program transformations, in particular of several optimisations of evaluation, including strictness optimisations.

We also analysed program transformations used in GHC w.r.t. the FUNDIO semantics. After turning off a few transformations which are not FUNDIO-correct, we have achieved a FUNDIO-compatible modification of GHC, which is called *HasFuse*. This compiler correctly compiles Haskell programs which make use of `unsafePerformIO` in the common (safe) sense, since problematic optimisations are turned off or performed more restrictively. HasFuse can also compile programs which make use of `unsafePerformIO` in arbitrary contexts, where the semantics is given by FUNDIO. This allows to combine `unsafePerformIO` with monadic I/O in Haskell, where the result is reliable in the sense that I/O operations will not astonishingly be duplicated.

Semantics and Transformations for Functional Hardware Descriptions

We investigated hardware descriptions in a functional language, i.e., Haskell programs extended by a parallel-or (`por`), where the non-deterministic operator `por` is implemented using Concurrent Haskell. As semantic model we use a call-by-need lambda calculus extended with `letrec`, `case`, constructors, and in particular with parallel-or. Ongoing research is devoted to prove correctness of circuit transformations, also including latches and combinational cycles, on the level of the high-level functional language descriptions.

Mutual Similarity and Finite Simulation

In order to achieve more inference rules for equality in call-by-need lambda-calculi, *Matthias Mann* has established a soundness (w.r.t. contextual equivalence) proof for mutual similarity in a non-deterministic call-by-need lambda calculus. Moreover, we have shown that this approach scales up well to more expressive call-by-need non-deterministic lambda calculi, i.e., similarity can be used as a co-induction-based proof tool for establishing contextual preorder in a large class of untyped higher-order call-by-need calculi, in particular calculi with constructors, `case`, `let`, and non-deterministic choice.

For non-deterministic call-by-need calculi with `letrec`, known approaches to prove that simulation implies contextual equivalence are inapplicable. A recent result obtained in collaboration with *Elena Machkasova* is correctness of a variation of simulation for checking and proving contextual equivalence in an extended non-deterministic call-by-need lambda-calculus with `letrec`. The basic technique for the simulation as well as the correctness proof is called pre-evaluation, which computes a set of answers for every closed expression. If simulation succeeds in finite computation depth, then it is guaranteed to show contextual preorder of expressions. Further research is to adapt and extend the methods to an appropriately defined simulation, and to investigate an extension of the tools and methods to a combination of may- and must-convergence.

Locally Bottom-Avoiding Choice

For modelling concurrent evaluation of functional programs, we investigated an extended call-by-need lambda-calculus with McCarthy's non-deterministic `amb`-operator. We introduced an observational equivalence based on may- and must-termination w.r.t. a fair small step reduction semantics. We proved correctness of several program transformations, in particular partial evaluation using deterministic reductions. We developed some nontrivial proof techniques including a standardisation theorem and a weak form of finite simulation for proving program equivalences. With these tools it appears promising to show correctness of further program transformations. As an implementation model we developed an abstract machine for lazy evaluation of concurrent computations, implemented a variant of this machine in Haskell, and proved correctness of this machine with respect to the calculus extended with `amb`.

Strictness Analysis using Abstract Reduction

The algorithm for strictness analysis using abstract reduction has been implemented at least twice: Once by Nöcker in C for Concurrent Clean and on the other hand by Schütz in Haskell in 1994. In 2005 we proved correctness of the algorithm by using a call-by-need lambda-calculus as a semantic basis. Most implementations of strictness analysis use set constants like $\top$ (all expressions) or $\bot$ (expressions that have no weak head normal form). We have shown that the subset relationship problem of co-inductively defined set constants is in DEXPTIME.

Further reading

- Chair for Artificial Intelligence and Software Technology: http://www.ki.informatik.uni-frankfurt.de
- References to all mentioned research topics are collected on the following webpage: http://www.ki.informatik.uni-frankfurt.de/research/HCAR.html

8.3 Functional Programming at the University of Kent

Report by: Olaf Chitil

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. We are seeking PhD students for funded research projects.

Our members pursue a variety of Haskell-related projects, many of which are reported in other sections of this report. Nik Sultana's MSc thesis on formal verification of Haskell refactorings has recently been accepted. Chris Brown continues extending HaRe, the Haskell Refactorer (→ 4.4.5). Thomas Davie and Olaf Chitil continued development of the Haskell tracer Hat (→ 4.3.5) and its theoretical foundations, studying in particular the representation of functional values as finite maps. This summer a student will work with Olaf Chitil on improving Heat and making a public release. Heat is a deliberately simple IDE for teaching Haskell that has been used at Kent for three years. Keith Hanna is continuing work on Vital, a document-centred programming environment for Haskell, and on

Pivotal, a GHC-based implementation of a similar environment. The Kent Systems Research Group is developing an occam compiler in Haskell (Tock). Neil Brown has created a Haskell library ("Communicating Haskell Processes") based on the Communicating Sequential Processes calculus.

Further reading

- FP group: http://www.cs.kent.ac.uk/research/groups/tcs/fp/
- Refactoring Functional Programs: http://www.cs.kent.ac.uk/projects/refactor-fp/
- Hat: http://www.haskell.org/hat/
- Vital: http://www.cs.kent.ac.uk/projects/vital/
- Pivotal: http://www.cs.kent.ac.uk/projects/pivotal/
- Tock: https://www.cs.kent.ac.uk/research/groups/sys/wiki/Tock

8.4 Foundations and Methods Group at Trinity College Dublin

Report by:	Andrew Butterfield
Participants:	Glenn Strong, Hugh Gibbons, Edsko de Vries

The Foundations and Methods Group focuses on formal methods, category theory, and functional programming as the obvious implementation method. A sub-group focuses on the use, semantics, and development of functional languages, covering such areas as:

- Supporting OO-Design technique for functional programmers
- Using functional programs as invariants in imperative programming
- Developing a GUI-based 2nd-order equational theorem prover (→ 6.5.4)
- New approaches to uniqueness typing, applicable to Hindley-Milner style type-inferencing (→ 3.3.1)

Recent work in this area included:

- Formal aspects of Functional I/O
- Using Testing to Debug Formal Models

Members of other research groups at TCD have also used Haskell, such as the work done on Image rendering using GHC/OpenGL, in the Interaction, Simulation, and Graphics Lab.

Further reading

https://www.cs.tcd.ie/research_groups/fmg/moin.cgi/FunctionalProgramming

8.5 Formal Methods at DFKI Lab Bremen and University of Bremen

Report by:	Christian Maeder
Participants:	Mihai Codescu, Dominik Lücke, Christoph Lüth, Christian Maeder, Achim Mahnke, Till Mossakowski, Lutz Schröder

The activities of our group centre on formal methods and the Common Algebraic Specification Language (CASL).

We are using Haskell to develop the Heterogeneous tool set (Hets), which consists of parsers (using Parsec), static analysers, and proof tools for languages from the CASL family, such as CASL itself, HasCASL, CoCASL, CspCASL, and ModalCASL, and additionally OMDoc and Haskell. 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 and HOLCF.

Another project using Haskell is the Proof General Kit, which designs and implements a component-based framework for interactive theorem proving. The central middleware of the toolkit is implemented in Haskell. The project is the successor of the highly successful Emacs-based Proof General interface. It is a cooperation of David Aspinall from the University of Edinburgh and Christoph Lüth from Bremen.

The Coalgebraic Logic Satisfiability Solver CoLoSS is being implemented jointly at DFKI-Lab Bremen and at the Department of Computing, Imperial College London. The tool is generic over representations of the syntax and semantics of certain modal logics; it uses the Haskell class mechanism, including multi-parameter type classes with functional dependencies, extensively to handle the generic aspects.

Other extensions, libraries, and tools of the Glasgow Haskell Compiler that we exploit include concurrency, existential and dynamic types, Template Haskell, DriFT, Haddock (→ 4.2.1), Programmatica, Shellac (→ 5.8.3), HaXml (→ 5.10.3), and hxt (→ 5.10.2). We also maintain old sources such as bindings to uDrawGraph (formerly Davinci) and Tcl/TK, Shared Annotated Terms (ATerms), and haifa-lite.

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
- Proof General Kit: http://proofgeneral.inf.ed.ac.uk/Kit
- The Coalgebraic Logic Satisfiability Solver CoLoSS: http://www.informatik.uni-bremen.de/~lschrode/projects/GenMod,

http://www.doc.ic.ac.uk/~dirk/COLOSS/

8.6 Functional Programming at Brooklyn College, City University of New York

Report by: Murray Gross

A grant has provided us with 6 new quad-processor machines, which we are currently integrating into our existing Linux/Mosix cluster. When the integration is complete, we will be comparing the performance and behaviour of the Brooklyn College version of GpH (→ 3.1.2) and the SMP facility of the latest release of GHC (→ 2.1).

In the area of applications, we are working on two AI projects, three-dimensional tic-tac-toe (noughts and crosses), and an extended version of the Sudoku puzzle. We have also begun work on a parallel implementation of Skibinski's quantum simulator, which we intend to use to study Grover's fast search algorithm.

Contact

Murray Gross ⟨magross@its.brooklyn.cuny.edu⟩

8.7 SCIence project

Report by: Kevin Hammond

SCIEnce (http://www.symbolic-computation.org/) is a 3.2M euros, 5-year project that brings together major developers of symbolic computing systems, including Maple, GAP, MuPAD, and Kant with the world-leading Centre for Research in Symbolic Computation at RISC-Linz, Austria.

It makes essential use of functional programming technology in the form of the GRID-GUM functional programming system for the Grid, which is built on the Glasgow Haskell Compiler. The objective is not the technically infeasible goal of rewriting all these (and more) complex systems in Haskell. Rather, we use GRID-GUM to link components built from each of the symbolic systems to form a coherent heterogeneous whole. In this way, we hope to achieve what is currently a pious dream for conventional Grid technology, and obtain a significant user base both for GRID-GUM and for Haskell. We are, of course, taking full advantage of Haskell's abilities to compose and link software components at a very high level of abstraction.

A fuller paper has appeared in the draft proceedings of the 2007 Symposium on Trends in Functional Programming (TFP 2007), New York, April 2007. A revised version is currently being prepared for submission to the post-symposium proceedings.

8.8 Bay Area Functional Programmers

Report by: Keith Fahlgren

The Bay Area Functional Programmers group held their inaugural meeting in September, giving programmers in the San Francisco Bay using or interested in functional programming and functional programming languages such as Haskell, OCaml, SML, Scala, and Erlang, a place to meet, discuss, and learn together. We followed up the first informal meeting with an October talk by Alex Jacobson on HAppS (http://happs.org/). November saw David Pollak present the Scala-based web framework lift (http://liftweb.net). We will finish our series on functional web frameworks with Yariv Sadan presenting the Erlang framework ErlyWeb (http://erlyweb.org/) in December. Videos & slides are available for all the talks at the BayFP blog, as well as information on how to join the mailing list: http://bayfp.org/blog.