Haskell Communities and Activities Report

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

Tenth Edition – June 12, 2006

Andres Löh (ed.)

Lloyd Allison, Tiago Miguel Laureano Alves, Krasimir Angelov
Dmitry Astapov, Alistair Bayley, Jean-Philippe Bernardy
Clifford Beshers, Edwin Brady, Björn Bringert
Chris Brown, Manuel Chakravarty, Olaf Chitil
Alain Crémieux, Iavor Diatchki, Atze Dijkstra
Robert Dockins, Frederik Eaton, Shae Erisson
Jan van Eijck, Martin Erwig, Sander Evers
Markus Forsberg, Simon Foster, Benjamin Franksen
Leif Frenzel, André Furtado, John Goerzen
Dimitry Golubovsky, Murray Gross, Walter Gussmann
Jurriaan Hage, Thomas Hallgren, Keith Hanna
Robert van Herk, Ralf Hinze, Paul Hudak
Liyang Hu, Graham Hutton, Johan Jeuring
Paul Johnson, Isaac Jones, Oleg Kiselyov
Marnix Klooster, Lemmih, Huiqing Li
Andres Löh, Rita Loogen, Salvador Lucas
Christoph Lüth, Ketil Malde, Christian Maeder
Simon Marlow, Paolo Martini, Conor McBride
Serge Mechveliani, Neil Mitchell, William Garret Mitchener
Andy Adams-Moran, J. Garrett Morris, Rickard Nilsson
Sven Panne, Ross Paterson, Jens Petersen
Simon Peyton-Jones, Bernie Pope, Claus Reinke
Frank Rosemeier, David Roundy, Alberto Ruiz
David Sabel, Tom Shackell, Uwe Schmidt
Martijn Schrage, Alexandra Silva, Axel Simon
Anthony Sloane, Dominic Steinitz, Donald Bruce Stewart
Martin Sulzmann, Doaitse Swierstra, Wouter Swierstra
Audrey Tang, Henning Thielemann, Peter Thiemann
Simon Thompson, Phil Trinder, Arjan van IJzendoorn
Miguel Vilaca, Joost Visser, Malcolm Wallace
Stefan Wehr, Ashley Yakeley, Bulat Ziganshin

Preface

This is the tenth edition of the Haskell Communities and Activities Report (HCAR) – a collection of entries about everything that is going on and related to Haskell in some way that appears twice a year. Perhaps the release of the tenth edition is a good time to have a look back at the beginning.

At the Haskell Workshop 2001, during the traditional discussion on "The Future of Haskell", Claus Reinke pointed out that it is very difficult to keep track of all the developments within the Haskell community: there are many people working on and with Haskell, they are located in very different places and are working on several different fields. He was asking for a collection of up-to-date information about releases of compilers, tools, and application, but also of ongoing work in a single place for everyone to look up.

Everyone agreed with Claus, but probably nothing would have happened if he himself had not volunteered to start the project. Already in November 2001, the first edition of the HCAR appeared. It had 20 contributors and the PostScript/PDF version consisted of 22 pages.

Four and a half years later, one can compare the current edition with the first and will notice that the basic structure has remained unchanged, but the size has nearly tripled (62 pages) and the number of contributors has almost reached a hundred (93 contributors). This demonstrates that the original concept was a good one, and that the HCAR is successful. I would therefore like to thank Claus for his work on the first five editions, Arthur van Leeuwen, the editor of the sixth edition, but most of all the countless contributors that have filled the report with content over the years. It is you that really make the report happen, and also a joy to work on, allowing me to read lots of interesting submissions while assembling the report.

I am very happy to see that the HCAR has quite some "competition" these days: the Haskell Weekly News (→ 1.4) gather Haskell-relevant information on a regular basis, the mailing lists are as active as they always are, the `#haskell` IRC channel (→ 1.3) is a direct contact to other people interested in Haskell and a reliable source of information about current work, the Haskell Sequence (→ 1.4.1) is a news portal for the Haskell world, the Haskell Planet (unfortunately without an entry) collects blogs by Haskellers, the Haskell Wiki has been integrated with the main haskell.org (→ 1.1) site, and I am sure I forgot a few other ways to get at information about Haskell communities and activities.

All the media mentioned above have different strengths and weaknesses, and therefore serve and reach different groups of people. It is good to see that there is so much communication in the Haskell world.

If you miss a project in this HCAR, then simply contribute to the next – either by writing an entry yourself, or by making the project leader aware of the HCAR! Please mark the final weeks of October in your calendar, because that is when the entries for the November edition will be collected.

As always, feedback is very welcome ⟨hcar@haskell.org⟩. Now, I wish you pleasant reading!

Andres Löh, University of Bonn, Germany

Contents

1 General

1.1 HaskellWiki and haskell.org

Report by: Ashley Yakeley

HaskellWiki is a MediaWiki installation now 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.org and Google Summer of Code 2006

Report by: Paolo Martini
Status: very active

Background

Google started to fund students working on Open Source/Free Software projects during the last summer through the Summer of Code programme. It aims to help students entering the FOSS development world, providing guidance and money for the summer period ($4,500 for three months of work.)

A number of organizations which runs active FOSS projects take part to this programme. They are required to provide mentors for students and publish a list of projects of interest. (More details can be found here: http://haskell.org/pipermail/haskell/2006-April/017872.html.)

The official Google site publishes detailed informations and deadlines, and it is located at http://code.google.com/soc/.

Status

haskell.org is officially a mentoring organization for this year's programme. Many community members volunteered for the organising and mentoring roles needed, an updated list of them can be found on the trac site we set up:

- http://hackage.haskell.org/trac/summer-of-code/

(The site also contains the list of projects proposals from the organization.)

The application period opened on May 1st, and ended on May 9th.

We got more than a hundred proposals from roughly 90 individual students!

Google has accepted nine of the proposals for funding – paid Haskell Open Source work for the summer:

- *Thin out cabal-get and integrate in GHC* by Paolo Martini, mentored by Isaac Jones
- *GHCi based debugger for Haskell* by José Iborra López, mentored by Lemmih
- *haskellnet* by Jun Mukai, mentored by Shae Matijs Erisson
- *A model for client-side scripts with HSP* by Joel Bjórnson, mentored by Niklas Broberg
- *Unicode ByteString, Data.Rope, Parsec for generic strings* by Spencer Janssen, mentored by Don Stewart
- *Port Haddock to use GHC* by David Waern, mentored by Simon Marlow
- *Fast Mutable Collection Types for Haskell* by Caio Marcelo de Oliveira Filho, mentored by Audrey Tang
- *Implement a better type checker for yhc* by Leon P Smith, mentored by Malcolm Wallace
- *Language.C – a C parser written in Haskell* by Marc Ernst Eddy van Woerkom, mentored by Manuel Chakravarty

The man in charge for the administrative work is Isaac Jones ⟨ijones@syntaxpolice.org⟩.

Further reading

- Students FAQ
  http://code.google.com/soc/studentfaq.html
- Mentors FAQ
  http://code.google.com/soc/mentorfaq.html
- Summer of Code 2005
  http://code.google.com/summerofcode05.html

1.3 #haskell

Report by: Shae Erisson

The `#haskell` IRC channel is a real-time text chat where anyone can join to discuss Haskell. `#haskell` averages about one hundred eighty users. Point your IRC client to irc.freenode.net and join the `#haskell` channel.

The `#haskell.se` channel is the same subject but discussion happens in Swedish. This channel tends to have a lot of members from Gothenburg.

There is also a `#darcs` channel – if you want real-time discussion about darcs (→ 6.6), drop by!

1.4 Haskell Weekly News

Report by: Don Stewart

The Haskell Weekly News (HWN) is a weekly newsletter covering developments in Haskell. Content includes announcements of new projects, discussions from the various Haskell communities, notable project commit messages, and more.

It is published in html form on The Haskell Sequence (→ 1.4.1), via mail on the Haskell mailing list, and via RSS. Headlines are published on haskell.org (→ 1.1).

Further reading

- Archives, and more information can be found at:
  http://www.haskell.org/haskellwiki/Haskell_Weekly_News

1.4.1 The Haskell Sequence

Report by: John Goerzen

The Haskell Sequence is a community-edited Haskell news and discussion site. Its main feature is a slashdot-like front page with stories and discussion about things going on in the Haskell community, polls, questions, or just observations. Submissions are voted on by the community before being posted on the front page, similar to Kuro5hin.

The Haskell Sequence also syndicates Haskell mailing list posts, Haskell-related blogs, and other RSS feeds in a single location. Free space for Haskell-related blogs, which require no voting before being posted, is also available to anyone.

Further reading

The Haskell Sequence is available at http://sequence.complete.org.

1.5 The Monad.Reader

Report by: Shae Erisson

There are plenty of academic papers about Haskell, and plenty of informative pages on the Haskell Wiki. But there's not much between the two extremes. The Monad.Reader aims to fit in there; more formal than a Wiki page, but less formal than a journal article.

Want to write about a tool or application that deserves more attention? Have a cunning hack that makes coding more fun? Got that visionary idea people should know about? Write an article for The Monad.Reader!

Further reading

See the TmrWiki for more information: http://www.haskell.org/tmrwiki/FrontPage.

1.6 Books and tutorials

1.6.1 "Hitchhickers Guide to Haskell" tutorial

Report by: Dmitry Astapov
Status: work in progress

"Hitchhickers Guide to Haskell" is a tutorial aimed to provide a "quick start into Haskell" for programmers with solid experience of other languages under their belt. Instead of "side by side" comparison between Haskell and another language of choice (like C or Java), the tutorial is built around case studies, which show how typical tasks are performed in Haskell.

This is work in progress, only 5 chapters have been written so far.

The tutorial is available on the Haskell wiki (URL below) or from the darcs repository at http://adept.linux.kiev.ua/repos/hhgtth.

Right now I am collecting ideas for subsequent chapters, so any feedback from readers is appreciated more than ever.

Further reading

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

1.6.2 New textbook – Programming in Haskell

Report by: Graham Hutton

The contract to publish the book with Cambridge University Press has recently been signed (including a clause about film rights, so expect "Haskell the Movie" in a couple of years `:-)`), and it is now entering the final production stage, with an estimated publication date in the last quarter of 2006. Further details, including a preview of the first five chapters and powerpoint lecture slides for all chapters, are available on the web from http://www.cs.nott.ac.uk/~gmh/book.html.

1.6.3 Haskell Tutorial WikiBook

Report by: Paul Johnson

The Haskell Tutorial WikiBook has made considerable progress over the last six months. It has now reached the stage where a beginner to Haskell should find it useful, although much still remains to be done.

Much of the recent work has been done by Eric Kow (Kowey). Jeff Newburn also gave permission for his "All About Monads" tutorial to be imported. Thanks to these, and to all the other contributors.

Future work needs to focus on:

- Adding exercises and worked examples. The first section in particular presents Haskell from the bottom up with very little to explain how the material relates to real world programming.
- Integrating the existing material. At present there are three distinct sections that read like three different tutorials, partly because that is how they started. All contributions are welcome.

Further reading

http://en.wikibooks.org/wiki/Programming:Haskell

2 Implementations

2.1 The Glasgow Haskell Compiler

Report by: Simon Peyton-Jones

We have been quite busy on GHC during the last six months. Here are the highlights:

- We switched to darcs (→ 6.6) for our version control. Thanks to all those who helped out, especially John Goerzen who migrated the repository over from CVS and helped with the initial setup.

  Following the darcs switchover, the GHC source tree has been reorganised and flattened, and the build system simplified.

- The multiprocessor support has had an overhaul. There are now per-CPU run queues, so thread affinity should be better and the scheduler is lock-free in the common case. There are some new RTS flags to control thread migration (although there is still no real load balancing). You don't need a separate set of libraries for running parallel code – the ordinary libraries work (this required a small performance hit for sequential code, unfortunately). The threaded and SMP runtimes were merged, so only `-threaded` is required for parallel now, `-smp` is a synonym for `-threaded`.

- Better GC behaviour for IORefs/STRefs, STArrays/IOArrays, and when there are a large number of blocked threads. Now these objects are only scanned during GC if they were mutated since the previous GC. This alleviates some pathological cases of poor GC performance, and gave GHC itself a performance boost (GHC uses IORefs for type variables). Large arrays are still scanned in their entirety; hopefully we'll improve this in the future.

- Refactoring in the compiler front-end (thanks to Lemmih).

- Loosen the rules for instance declarations (thanks to Ross Paterson).

  http://www.haskell.org//pipermail/glasgow-haskell-users/2006-February/009633.html

- Implementation of "boxy types" to support type inference for impredicative types:

  http://research.microsoft.com/~simonpj/papers/boxy

  Huge thanks to Stephanie Weirich and Dimitrios Vytiniotis for their work on this project.

- The design of lexically-scoped type variables has changed, and will likely change further. These changes have arisen out of the work with Stephanie and Dimitrios; frankly we aren't yet sure what the exact design should be and, until we are, the implementation has rough edges.

- Name completion in GHCi. If you have readline, then you can use the TAB key on the command line to complete against names in scope.

- Full support for Unicode (UTF-8) source files. UTF-8 is currently the only encoding supported by GHC, we plan to add support for more encodings before 6.6 is released.

- Bang patterns are now implemented, as an experimental feature

  http://haskell.galois.com/cgi-bin/haskell-prime/trac.cgi/wiki/BangPatterns

- The `Data.ByteString` (→ 4.6.3) library was added (thanks particularly to Don Stewart and others who contributed to the code).

- The native code generator can handle loops (thanks Wolfgang Thaller).

- Performance improvements for `x86_64`: more arguments are passed in registers.

- Experimental "breakpoint" feature in GHCi (thanks Lemmih).

Release cycle

- 6.4.2 was released recently, with a significant number of bugfixes relative to 6.4.1. Nevertheless, problems have been found in 6.4.2; there is an unidentified bug affecting MacOS X, and problems have been reported in the threaded RTS on Solaris and FreeBSD (in fact, these problems may have been present in 6.4.1 but unnoticed until now, because in 6.4.2 GHC itself was switched to the threaded RTS).

- There will probably be a 6.4.3, with a few important fixes only.

- 6.6, with all the new features above, is scheduled for sometime this summer. To get an idea of what is still to do for 6.6, check the ticket system on the Trac:

  http://hackage.haskell.org/trac/ghc/query?status=new&status=assigned&status=reopened&milestone=6.6&order=priority

  Help is welcome as usual, and will probably speed the release along!

Forthcoming excitements

- We are working hard on developing a data-parallel extension to GHC, in collaboration with Gabriele Keller, Manuel Chakravarty, and Roman Leshchinskiy. The basic idea was pioneered by Guy Blelloch in NESL, and subsequently developed in various ways by Gabi, Manuel, and Roman; for example, see http://www.cse.unsw.edu.au/~chak/papers/CKLP01.html
- We plan to implement Associated Types during the next few months: http://research.microsoft.com/~simonpj/papers/assoc-types/
- Part of the reason that we have been slow to implement associated types is that they can't be translated into System F (GHC's internal language without great difficulty). We have recently figured out a new design for GHC's internal language, which we call System FC:

  http://research.microsoft.com/~simonpj/papers/ext-f

  Kevin Donnelly is spending the summer at Microsoft as an intern, to implement FC in GHC.
- The multiprocessor GHC runs Haskell in parallel, but when garbage collection happens only one processor does the job. How embarrassing. Roshan James, also an intern this summer, is going to build a parallel garbage collector.

We are planning to run a "GHC Hackathon" just before ICFP in Portland, in September 2006. We'll give some tutorials on how GHC works inside, and then spend some time writing code together. If you are interested, keep an eye on GHC's web site.

2.2 Hugs

Report by:	Ross Paterson
Status:	stable, actively maintained, volunteers welcome

The May 2006 release of Hugs was the first for over three years to include a Windows distribution, thanks to Neil Mitchell, who has also contributed a new version of WinHugs, the Windows graphical interface to Hugs. Other major features, already seen in the Unix-only release, are support for the Cabal infrastructure (→ 4.1.1), Unicode support (contributed by Dmitry Golubovsky) and lots of up-to-date libraries.

Obsolete non-hierarchical libraries (hslibs and Hugs-specific libraries) have been removed from the default search path, and will disappear altogether in the next release.

The source distribution is available in two forms: a huge omnibus bundle containing the Hugs programs and lots of useful libraries, or a minimal bundle, with most of the libraries hived off as separate Cabal packages.

There is also a new bug tracking system and development wiki at http://hackage.haskell.org/trac/hugs.

As ever, volunteers are welcome.

2.3 nhc98

Report by:	Malcolm Wallace
Status:	stable, maintained

nhc98 is a small, easy to install, compiler for Haskell'98. It is in stable maintenance-only mode – the current public release is version 1.18. Maintenance continues in CVS at haskell.org. Recent news is that Niklas (the original author of nhc13) has contributed a workaround for the hi-mem bug which was preventing the compiler from building on many recent Linux platforms. A new release incorporating this patch is expected soon.

The Yhc (→ 2.4) branch of nhc98 is also making good progress.

Further reading

http://haskell.org/nhc98

2.4 yhc

Report by:	Tom Shackell
Status:	work in progress

The York Haskell Compiler (yhc) is a backend rewrite of the nhc98 (→ 2.3) compiler to support features such as a platform independent bytecode and runtime system.

It is currently work in progress, it compiles and correctly runs almost every standard Haskell 98 program but FFI support is on going. Contributions are welcome.

Further reading

- Homepage:
  http://www.cs.york.ac.uk/~ndm/yhc/
- Darcs (→ 6.6) repository:
  http://www.cs.york.ac.uk/fp/darcs/yhc/

3 Language

3.1 Variations of Haskell

3.1.1 Haskell on handheld devices

Report by:	Anthony Sloane
Status:	unreleased

Our work on running Haskell on handheld devices based on Palm OS has taken a different direction since the last report. Instead of basing our port on nhc98 (→ 2.3) we are now using yhc (→ 2.4) as the basis. Overall everything is easier since yhc has a better separation of compiler and runtime system. We have also added a part-time programmer to this project so more progress is being made. An alpha version is close to working.

3.1.2 Vital: Visual Interactive Programming

Report by:	Keith Hanna
Status:	stable (latest release: April 2005)

Vital is a highly interactive, visual environment that aims to present Haskell in a form suitable for use by engineers, mathematicians, analysts and other end users who often need a combination of the expressiveness and robustness that Haskell provides together with the ease of use of a 'liveŠ graphical environment in which programs can be incrementally developed.

In Vital, Haskell modules are presented as 'documentsŠ having a free-form layout and with expressions and their values displayed together. These values can be displayed either textually, or pictorially and can be manipulated by an end user by point-and-click mouse operations. The way that values of a given type are displayed and the set of editing operations defined on them (i.e., the 'look and feel' of the type) are defined using type classes. For example, an ADT representing directed graphs could be introduced, with its values displayed pictorially as actual directed graphs and with the end user provided with a menu of operations allowing edges to be added or removed, transitive closures to be computed, etc. (In fact, although an end user appears to be operating directly on values, it is actually the Haskell program itself that is updated by the system, using a specialised form of reflection.)

The present implementation includes a collection of interactive tutorial documents (including examples illustrating approaches to exact real arithmetic, pictorial manipulation of DNA and the genetic code, animated diagrams of mechanisms, and the composition and synthesis of MIDI music).

The Vital system can be run via the web: a single mouse-click is all that is needed!

Further reading

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

3.1.3 Pivotal: Visual Interactive Programming

Report by:	Keith Hanna
Status:	active (first release: November 2005)

Pivotal 0.025 is a very early prototype of a Vital-like environment (→ 3.1.2) for Haskell. Unlike Vital, however, Pivotal is implemented entirely in Haskell. The implementation is based on the use of the hs-plugins library (→ 4.4.1) to allow dynamic compilation and evaluation of Haskell expressions together with the gtk2hs library (→ 4.8.1) for implementing the GUI.

At present, the implementation is only in a skeletal state but, nevertheless, it provides some useful functionality. The Pivotal web site provides an overview of its principles of operation, a selection of screen shots (including a section illustrating image transforms in the complex plane), and a (very preliminary!) release of the Haskell code for the system.

A more extensive implementation (based on the use of the GHC API (→ 2.1) for reflection, in place of the hs-plugins (→ 4.4.1) mechanism) is planned as soon as the required hooks are available in GHC 6.6.

Further reading

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

3.1.4 House (formerly hOp)

Report by:	Thomas Hallgren
Status:	active development

House is a platform for exploring various ideas relating to low-level and system-level programming in a high-level functional language, or in short for building operating systems in Haskell. House is based on hOp by Sébastien Carlier and Jérémy Bobbio.

Recent work includes

- the introduction of H, the Hardware Monad, an API on top of which various operating system features (e.g., virtual memory management, user-space execution, device drivers and interrupt handling) can be implemented in a fairly safe way. Key properties of

the H monad operations are captured as P-Logic assertions in the code. This is described in more detail in our ICFP 2005 paper.

The House demo system is now implemented on top of the H monad. There is also work in progress on implementing an L4 compatible micro-kernel on top of H.

- adding support for parsing and rendering GIF images. This allowed us to use House to display the slides for the talk at ICFP.
- adding support for scanning the PCI bus and identifying PCI devices.

Further reading

Further information, papers, source code, demos and screenshots are available here: http://www.cse.ogi.edu/~hallgren/House/

3.1.5 Camila

Report by:	Alexandra Silva and Joost Visser

The Camila project explores how concepts from the VDM++ specification language and the functional programming language Haskell can be combined. On the one hand, it includes experiments of expressing VDM's data types (e.g. maps, sets, sequences), data type invariants, pre- and post-conditions, and such within the Haskell language. On the other hand, it includes the translation of VDM specifications into Haskell programs. Moreover, the use of the OOHaskell library (→ 4.6.7) allows the definition of classes and objects and enables important features such as inheritance. In the near future, support for parallelism and automatic translation of VDM++ specifications into Haskell will be added to the libraries.

Currently, the project has produced first versions of the Camila Library, both distributed as part of the UMinho Haskell Libraries and Tools. The library resorts to Haskell's constructor class mechanism, and its support for monads and monad transformers to model VDM's datatype invariants, and pre- and post-conditions. It allows switching between different modes of evaluation (e.g. with or without property checking) by simply coercing user defined functions and operations to different specific types.

Further reading

The web site of Camila (http://wiki.di.uminho.pt/wiki/bin/view/PURe/Camila) provides documentation. Both library and tool are distributed as part of the UMinho Haskell Libraries and Tools.

3.1.6 HASP

Report by:	Lemmih
Status:	active

HASP is a fork of Niklas Broberg's Haskell Server Pages. Changes includes:

- support for all GHC extensions
- use of the GHC-api (→ 2.1) for byte-code compilations
- front-end based on FastCGI instead of its own web server
- minor bug fixes and performance tuning.

Some of the features implemented in HASP will be ported back into the main HSP tree. However, experimental features like byte code generation via the GHC api will most likely stay in HASP.

Further reading

- Darcs repository: http://darcs.haskell.org/~lemmih/hasp/
- Original HSP: http://www.cs.chalmers.se/~d00nibro/hsp/

3.2 Non-sequential Programming

3.2.1 Data Parallel Haskell

Report by:	Manuel Chakravarty
Status:	active

Data Parallel Haskell is the codename for an extension to the Glasgow Haskell Compiler and its libraries to support nested data parallelism with a focus to utilise multi-core CPUs and other SMP hardware. The project is still in its early stages. For more information and code see http://www.cse.unsw.edu.au/~chak/project/dph/.

3.2.2 GpH – Glasgow Parallel Haskell

Report by:	Phil Trinder
Participants:	Phil Trinder, Abyd Al Zain, Greg Michaelson, Kevin Hammond, Yang Yang, Jost Berthold, Murray Gross

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

- We have developed an adaptive runtime environment (GRID-GUM) for GpH on computational grids. GRID-GUM incorporates new load management mechanisms that cheaply and effectively combine static and dynamic information to adapt to the heterogeneous and high-latency environment of a multi-cluster computational grid. We have made comparative measures of GRID-GUM's performance on high/low latency grids and heterogeneous/homogeneous grids using clusters located in Edinburgh, Munich and Galashiels. Results are published in:

  Al Zain A. *Implementing High-Level Parallelism on Computational Grids*, PhD Thesis, Heriot-Watt University, 2006.

  Al Zain A. Trinder P.W. Loidl H.W. Michaelson G.J. *Managing Heterogeneity in a Grid Parallel Haskell*, Journal of Scalable Computing: Practice and Experience 7(3), (September 2006).

- The design of a generic parallel runtime environment encompassing both the Eden and GpH runtime environments is complete, but the implementation is stalled at present.

- SMP-GHC, an implementation of GpH for multi-core machines has been developed by Tim Harris, Simon Marlow and Simon Peyton Jones (→ 2.1).

- At St Andrews GpH is being used as a vehicle for investigating scheduling on the GRID.

- We are teaching parallelism to undergraduates using GpH at Heriot-Watt and Phillips Universitat Marburg.

GpH Applications

- GpH is being used to parallelise the GAP mathematical library in an EPSRC project (GR/R91298).

- As part of the SCIEnce EU FP6 I3 project (026133) that started in April 2006 we will use GpH and Java to provide access to Grid services from Computer Algebra(CA) systems, including GAP and Maple. We will both produce Grid-parallel implementations of common CA library functions, and also wrap CA systems as Grid services.

Implementations

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

- 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.

- The unstable branch (GUM-5.02, based on GHC-5.02) is currently being tested on a Beowulf cluster. The unstable branch is available from the GHC CVS repository via tag gum-5-02-3.

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

- A port to a Sun-Solaris shared-memory machine exists but currently suffers from performance problems.

- A port to a Mosix cluster has been built in the Metis project at Brooklyn College, with a first version available on request from Murray Gross (→ 7.3.4).

Further reading

- GpH Home Page: 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⟩

3.2.3 GdH – Glasgow Distributed Haskell

Report by:	Phil Trinder
Participants:	Phil Trinder, Hans-Wolfgang Loidl, Jan Henry Nyström, Robert Pointon

GdH supports distributed stateful interactions on multiple locations. It is a conservative extension of both Concurrent Haskell and GpH (→ 3.2.2), enabling the distribution of the stateful IO threads of the former on the multiple locations of the latter. The programming model includes forking stateful threads on remote locations, explicit communication over channels, and distributed exception handling.

Status

An alpha-release of the GdH implementation is available on request ⟨gph@macs.hw.ac.uk⟩. It shares substantial components of the GUM implementation of GpH (Glasgow parallel Haskell) (→ 3.2.2).

Applications and Evaluation

- EPSRC project *High Level Techniques for Distributed Telecommunications Software* (GR/R88137) has recently been completed (February 2006). The project was collaboration between Heriot-Watt University and Motorola UK Research Labs, and

amongst other activities evaluated GdH and Erlang as technologies for distributed telecoms software in comparison to C++/CORBA. Previous publications appear on the project page, and some recent results are given in the papers below. The latter paper compares Erlang, GdH and C++ for engineering a medium-scale (14K lines of C++, 4K lines of Erlang, and 0.5K lines of GdH) telecoms component.

Nystrom J.H. Trinder P.W. King D.J. *Are High-level Languages suitable for Robust Telecoms Software?* Proc. 24th Int. Conference on Computer Safety, Reliability and Security (SAFECOMP'05), Fredrikstad, Norway (September 2005).

http://www.macs.hw.ac.uk/~dsg/telecoms/publications/SafeComp2005.pdf

Nystrom, J.H., Trinder, P.W., King,D.J. *A Comparative Evaluation of Three High-level Distributed Languages for Telecoms Software.* In preparation.

- There is a forthcoming Ph.D. thesis on the design, implementation and use of GdH by Robert Pointon.

Further reading

- The GdH homepage: http://www.macs.hw.ac.uk/~dsg/gdh/

3.2.4 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, Clara Segura

in Marburg: Rita Loogen, Jost Berthold, Steffen Priebe

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.

Eden has been implemented by modifying the parallel runtime system GUM of GpH (→ 3.2.2). Differences include stepping back from a global heap to a set of local heaps to reduce system message traffic and to avoid global garbage collection. The current (freely available) implementation is based on GHC 5.02.3. A source code version is available from the Eden web page. Installation support will be provided if required.

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 (Special Issue on Functional Approaches to High-Performance Parallel Programming)

Recent and Forthcoming Publications

- Jost Berthold, Rita Loogen: *The Impact of Dynamic Channels on Functional Topology Skeletons*, Parallel Processing Letters, to appear 2006.

- Steffen Priebe: *Dynamic Task Generation and Transformation within a Nestable Workpool Skeleton*, Euro-Par 2006, Dresden, to appear.

Current Activities

- Yolanda and Mercedes analyse Eden skeletons using an implementation of its operational semantics in Maude.

- Jost continues his work on a more general implementation of parallel Haskell dialects in a shared runtime system.

- Steffen continues his work on the polytypic skeleton library for Eden making use of the new metaprogramming facilities in GHC.

- Jost and Rita continue working on the skeleton library.

Further reading

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

3.3 Type System/Program Analysis

3.3.1 Epigram

Report by: Conor McBride and Wouter Swierstra

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)).

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 won't 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.

Current Status

Whilst at Durham, Conor McBride developed the Epigram prototype in Haskell, interfacing with the xemacs editor. Nowadays, a team of willing workers at the University of Nottingham are developing a new version of Epigram, incorporating both significant improvements over the previous version and experimental features subject to active research.

The first steps have been made in collecting recurrent programs and examples in some sort of standard library. There's still a great deal of cleaning up to do, but progress is being made.

The Epigram system has also been used succesfully by Thorsten Altenkirch in his undergraduate course on Computer Aided Formal Reasoning for two years http://www.cs.nott.ac.uk/~txa/g5bcfr/. Several final year students have successfully completed projects that involved both new applications of and useful contributions to Epigram.

Peter Morris is working on how to build the datatype system of Epigram from a universe of containers. This technology would enable datatype generic programming from the ground up. Central to these ideas is the concept of *indexed container* that has been developed recently. There are ongoing efforts to elaborate the ideas in Edwin Brady's PhD thesis about efficiently compiling dependently typed programming languages.

Joel Wright has started writing a stand alone editor for Epigram using Gtk2Hs (→ 4.8.1). Thanks to a most helpful visit from Duncan Coutts and Axel Simon, two leading Gtk2Hs developers, we now have the beginnings of a structure editor for Epigram 2.

There has also been steady progress on Epigram 2 itself. The type theoretic basis underpinning Epigram has been further developed to incorporate *observational type theory*. The lion's share of the core theory has already been implemented, but there is still plenty of work to do.

Whilst Epigram seeks to open new possibilities for the future of strongly typed functional programming, its implementation benefits considerably from the present state of the art. Our implementation makes considerable use of applicative functors, higher-kind polymorphism and type classes. Moreover, its denotational approach translates Epigram's lambda-calculus directly into Haskell's. On a more practical note, we have recently shifted to the darcs version control system and cabal framework.

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 communicate via the mailing list ⟨epigram@durham.ac.uk⟩. The current implementation is naive in design and slow in practice, but it is adequate to exhibit small examples of Epigram's possibilities. The new implementation, whose progress can be observed at http://www.e-pig.org/epilogue/ will be much less rudimentary.

3.3.2 Chameleon project

Report by: Martin Sulzmann

Chameleon is a Haskell style language which integrates sophisticated reasoning capabilities into a program-

ming language via its CHR programmable type system. Thus, we can program novel type system applications in terms of CHRs which previously required special-purpose systems.

Latest developments

Jeremy Wazny successfully defended his PhD thesis on *Type inference and type error diagnosis for Hindley/Milner with Extensions*. This thesis summarizes to a large extent the theoretical underpinnings behind Chameleon. A copy can be downloaded via http://www.comp.nus.edu.sg/~sulzmann/chameleon/.

The latest available Chameleon version is from July 2005. This version is known to have bugs. We are currently working on a much improved version which will be available in the third quarter of 2006. An announcement will be sent to the Haskell mailing list once the new version is ready.

Further reading

http://www.comp.nus.edu.sg/~sulzmann/chameleon/

3.3.3 XHaskell project

Report by:	Martin Sulzmann
Participants:	Kenny Zhuo Ming Lu and Martin Sulzmann

XHaskell is an extension of Haskell with XDuce style regular expression types and regular expression pattern matching. We have much improved the implementation which can found under the XHaskell home-page.

Further reading

http://www.comp.nus.edu.sg/~luzm/xhaskell/

3.3.4 Constraint Based Type Inferencing at Utrecht

Report by:	Jurriaan Hage
Participants:	Bastiaan Heeren, Jurriaan Hage, Doaitse Swierstra

With the generation of understandable type error messages in mind we have devised a constraint based type inference method in the form of the Top library. This library is used in the Helium compiler (for learning Haskell) developed at Universiteit Utrecht. Our philopsophy is that no single type inferencer works best for everybody all the time. Hence, we want a type inferencer adaptable to the programmer's needs without the need for him to delve into the compiler. Our goal is to devise a library which helps compiler builders add this kind of technology to their compiler.

The main outcome of our work is the Top library which has the following characteristics:

- It uses constraints to build a constraint tree which follows the shape of the abstract syntax tree.
- These constraints can be ordered in various ways into a list of constraints
- Various solvers (specifically a fast greedy one, a slower global one, and the chunky solver which combines the two) exist to solve the resulting list of constraints.
- The library is easily extended with new constraints, and the type graph implementation includes various heuristics to find out what is the most likely source of an inconsistency. Some of these heuristics are very general, others are more tailored towards Haskell. Some the heuristics are fixed, like a majority heuristics which takes into account that there is 'more' evidence that a certain constraint is the root of an inconsistency. In addition, there are also heuristics specified from the outside. By means of a siblings directive, a programmer may specify that his experiences are that certain functions are often mixed up. As a result, a compiler may give the hint that (++) should be used instead of (:), because (++) happens to fit in the context.
- It preserves type synonyms as much as possible,
- We have support for type class directives. It allows programmers to for instance specify that certain instances will never occur. The type inferencer can use this information to give better error messages. Other directives can be used to specify additional invariants on type classes. For instance, that two type classes do not share a common type (Fractional vs. Integral). A paper about this subject will find its way into PADL 2005. Although we have implemented this into Helium, the infrastructure applies as well to other systems of qualified types.
- The various phases in type inferencing have now been integrated by a slightly different, more general choice of constraints.

An older version of the underlying machinery for the type inferencer has been published in the Proceedings of the Workshop of Immediate Applications of Constraint Programming held in October 2003 in Kinsale, Ireland.

The entire library is parameterized in the sense that for a given compiler we can choose which information we want to drag around.

The library has been used extensively in the Helium compiler, so that Helium can be seen as a case study in applying Top in a real compiler. In addition to the above, Helium also

- has a logging facility for building collections of correct and incorrect Haskell programs (including time line information),

- has a run-time parameters for experimenting with various solvers and constraint orderings.

- gives precise error location information,

- supports specialized type rules, which are a means to override the order in which certain expressions are inferenced and how the type error messages are formulated (see our paper presented at ICFP '03). These type rules are especially useful for making the type error messages for domain specific extensions to Haskell correspond more closely to the domain, instead of the underlying Haskell language structures. The specialized type rules are automatically checked for soundness and completeness with respect to the original type system.

Further reading

- Project website:
  http://www.cs.uu.nl/wiki/Top/WebHome

3.3.5 EHC, 'Essential Haskell' Compiler

Report by:	Atze Dijkstra
Participants:	Atze Dijkstra, Doaitse Swierstra
Status:	active development

The purpose of the EHC project is to provide a description of a Haskell compiler which is as understandable as possible so it can be used for education as well as research.

For its description an Attribute Grammar system (AG) is used as well as other formalisms allowing compact notation like parser combinators. For the description of type rules, and the generation of an AG implementation for those type rules, we recently started using the Ruler system (→ 5.5.3) (included in the EHC project).

The EHC project also tackles other issues:

- In order to avoid overwhelming the innocent reader, the description of the compiler is organised as a series of increasingly complex steps. Each step corresponds to a Haskell subset which itself is an extension of the previous step. The first step starts with the essentials, namely typed lambda calculus.

- Each step corresponds to an actual, that is, an executable compiler. Each of these compilers is a compiler in its own right so experimenting can be done in isolation of additional complexity introduced in later steps.

- The description of the compiler uses code fragments which are retrieved from the source code of the compilers. In this way the description and source code are kept synchronized.

Currently EHC already incorporates more advanced features like higher-ranked polymorphism, partial type signatures, class system, explicit passing of implicit parameters (i.e. class instances), extensible records, kind polymorphism.

Part of the description of the series of EH compilers is available as a PhD thesis, which incorporates previously published material on the EHC project.

The compiler is used for small student projects as well as larger experiments such as the incorporation of an Attribute Grammar system.

We also hope to provide a Haskell frontend dealing with all Haskell syntactic sugar omitted from EHC.

Further reading

- Homepage:
  http://www.cs.uu.nl/groups/ST/Ehc/WebHome
- Attribute grammar system:
  http://www.cs.uu.nl/wiki/HUT/AttributeGrammarSystem
- Parser combinators:
  http://www.cs.uu.nl/wiki/HUT/ParserCombinators

3.4 Generic Programming

Report by:	Johan Jeuring

Software development often consists of designing a (set of mutually recursive) datatype(s), to which functionality is added. Some functionality is datatype specific, other functionality is defined on almost all datatypes, and only depends on the type structure of the datatype.

Examples of generic (or polytypic) functionality defined on almost all datatypes are the functions that can be derived in Haskell using the deriving construct, storing a value in a database, editing a value, comparing two values for equality, pretty-printing a value, etc. Another kind of generic function is a function that traverses its argument, and only performs an action at a small part of its argument. A function that works on many datatypes is called a generic function.

There are at least two approaches to generic programming: use a preprocessor to generate instances of generic functions on some given datatypes, or extend a programming language with the possibility to define generic functions. The techniques behind some of these ideas are given in a separate subsection. In *Comparing approaches to generic programming in Haskell* (in the lecture notes of the Spring School on Datatype-Generic Programming 2006, held in Nottingham, April 2006, to appear in LNCS), Ralf Hinze, Johan Jeuring and Andres Löh compare 8 different approaches to generic programming in Haskell, both lightweight approaches and language extensions. Most of the approaches discussed

in this and previous versions of the Communities report are addressed. In the same set of lecture notes, Jeremy Gibbons discusses the various interpretations of the word 'generic'.

Preprocessors

DrIFT is a preprocessor which generates instances of generic functions. It is used in Strafunski (→ 4.3.2) to generate a framework for generic programming on terms. New releases appear regularly, the latest is 2.2.0 from April 2006.

Languages

Light-weight generic programming There are a number of approaches to light-weight generic programming.

Generic functions for data type traversals can (almost) be written in Haskell itself (using many of the extensions of Haskell provided by GHC), as shown by Ralf Lämmel and Simon Peyton Jones in the 'Scrap your boilerplate' (SYB) approach (http://www.cs.vu.nl/boilerplate/). The SYB approach to generic programming in Haskell has been further elaborated in the recently published (in FLOPS '06) paper *"Scrap Your Boilerplate" Reloaded* and *"Scrap Your Boilerplate" Revolutions* (to appear in MPC'06). In these papers Ralf Hinze, Andres Löh, and Bruno Oliveira show, amongst others, how by viewing the SYB approach in a particular way, the choice of basic operators becomes obvious.

In *Open data types and open functions* (to appear at PPDP'06), Andres Löh and Ralf Hinze propose to add extensible data types to Haskell, and they show how to use these extensible data types to implement generic functions in a light-weight approach to generic programming.

In *Generics as a Library*, Bruno Oliveira, Ralf Hinze and Andres Löh show how to extend Ralf Hinze's "Generic for the Masses" approach to be able to extend generic functions with ad-hoc behaviour for new data types.

Finally, in *Generic programming, NOW!* (in the lecture notes of the Spring School on Datatype-Generic Programming 2006, held in Nottingham, April 2006, to appear in LNCS), Ralf Hinze and Andres Löh show how GADTs can be used to implement many of the lightweight approaches to generic programming directly in Haskell.

Generic Haskell In *Generic views on data types* (to appear in MPC'06) Stefan Holdermans, Johan Jeuring, Andres Löh, and Alexey Rodriguez show how to add views on data types to Generic Haskell. Using these views, typical fixed-point functions such as determining the recursive children of a constructor of a recursive data type can be combined with the usual Generic Haskell programs in a single program. The Generic Haskell compiler has been extended with views (available via svn).

Other In *Generic Programming with Sized Types* (to appear in MPC'06), Andreas Abel defines a generic programming language in which you can only define terminating generic programs, by adding sizes to types.

In *iData for the World Wide Web: programming interconnected web forms* (in FLOPS'06), Rinus Plasmeijer and Peter Achten show how to use the generic programming extension of Clean for implementing web forms.

Techniques

Jeremy Gibbons' tutorial *Design Patterns as Higher-Order Datatype-Generic Programs* from ECOOP and OOPSLA 2005 has been written up as a paper, http://www.comlab.ox.ac.uk/jeremy.gibbons/publications/#hodgp. He and Bruno Oliveira have also written about *The Essence of the Iterator Pattern* as a higher-order datatype-generic program (http://www.comlab.ox.ac.uk/jeremy.gibbons/publications/#iterator), in terms of McBride and Paterson's idioms or applicative functors.

The Spring School on Datatype-Generic Programming has taken place in Nottingham, UK, April 23 - 26, see http://www.cs.nott.ac.uk/ssdgp2006/. There were lectures about comparing approaches to generic programming in Haskell, generic programming in Haskell using GADTs, the implementation of patterns as generic programs, generic programming in Omega (a Haskell-like functional programming language with a limited form of dependent types), and in Epigram (→ 3.3.1) (a dependently typed programming language).

Current Hot Topics

Generic Haskell: finding transformations between data types. Adding type inference to the compiler. Other: the relation between generic programming and dependently typed programming; the relation between coherence and generic programming; methods for constructing generic programs. Methods for testing generic programs.

Hopefully there will be papers about these topics in the next Workshop on Generic Programming (colocated with ICFP 2006): http://www.informatik.uni-bonn.de/~ralf/wgp2006.html.

Further reading

- http://repetae.net/john/computer/haskell/DrIFT/
- http://www.cs.chalmers.se/~patrikj/poly/
- http://www.generic-haskell.org/
- http://www.cs.vu.nl/Strafunski/
- http://www.cs.vu.nl/boilerplate/

4 Libraries

4.1 Packaging and Distribution

4.1.1 Hackage and Cabal

Report by:	Isaac Jones

Background

The Haskell Cabal is a Common Architecture for Building Applications and Libraries. It is an API distributed with GHC (→ 2.1), NHC98 (→ 2.3), and Hugs (→ 2.2) which allows a developer to easily group together a set of modules into a package.

HackageDB (Haskell Package Database) is an online database of packages which can be interactively queried by client-side software such as the prototype cabal-get. From HackageDB, an end-user can download and install packages which conform to the Cabal interface.

The Haskell Implementations come with a good set of standard libraries included, but this set is constantly growing and is maintained centrally. This model does not scale up well, and as Haskell grows in acceptance, the quality and quantity of available libraries is becoming a major issue.

It can be very difficult for an end user to manage a wide variety of dependencies between various libraries, tools, and Haskell implementations, and to build all the necessary software at the correct version numbers on their platform: previously, there was no generic build system to abstract away differences between Haskell Implementations and operating systems.

HackageDB and The Haskell Cabal seek to provide some relief to this situation by building tools to assist developers, end users, and operating system distributors.

Such tools include a common build system, a packaging system which is understood by all of the Haskell Implementations, an API for querying the packaging system, and miscellaneous utilities, both for programmers and end users, for managing Haskell software.

Further reading

- http://www.haskell.org/cabal
- http://hackage.haskell.org/ModHackage/Hackage.hs?action=home

4.2 General libraries

4.2.1 Hacanon-light

Report by:	Lemmih
Status:	usable, unmaintained

Hacanon-light is a lightweight FFI library that uses the Data Interface Scheme (DIS) from Hacanon (http://haskell.org/hawiki/Hacanon) and Template Haskell to provide a high level interface to marshaling/unmarshaling. It differs from Hacanon taking a passive role in the binding process; it won't use or validate itself from any foreign header files.

Hacanon-light is meant to be used together with Zeroth (→ 5.5.2).

Further reading

- Darcs repository:
  http://darcs.haskell.org/~lemmih/hacanon-light

4.2.2 HODE

Report by:	Lemmih
Status:	usable, unmaintained

HODE is a binding to the Open Dynamics Engine. ODE is an open source, high performance library for simulating rigid body dynamics.

HODE uses Hacanon-light (→ 4.2.1) to simplify the binding process and Zeroth (→ 5.5.2) to avoid linking with Template Haskell.

Further reading

- Darcs repository:
  http://darcs.haskell.org/~lemmih/hode
- ODE:
  http://ode.org

4.2.3 PFP – Probabilistic Functional Programming Library for Haskell

Report by:	Martin Erwig
Status:	active development

The PFP library is a collection of modules for Haskell that facilitates probabilistic functional programming, that is, programming with stochastic values. The probabilistic functional programming approach is based on a data type for representing distributions. A distribution represent the outcome of a probabilistic event

as a collection of all possible values, tagged with their likelihood.

A nice aspect of this system is that simulations can be specified independently from their method of execution. That is, we can either fully simulate or randomize any simulation without altering the code which defines it.

The library was developed as part of a simulation project with biologists and genome researchers. We plan to apply the library to more examples in this area. Future versions will hopefully contain a more systematically documented list of examples.

Since the last report, the implementation has undergone only minor changes, and no new release has been made yet. The web site has been updated slightly and contains the latest papers on the subject.

Further reading

http://eecs.oregonstate.edu/~erwig/pfp/

4.2.4 Hmm: Haskell Metamath module

Report by:	Marnix Klooster
Status:	Hmm 0.1 released, slow-paced development

Hmm is a small Haskell library to parse and verify Metamath databases.

Metamath (http://metamath.org) was conceived and almost completely implemented by Norman Megill. It a project for formalizing mathematics, a file format for specifying machine-checkable proofs, and a program for generating and verifying this file format. Already more than 6000 proofs have been verified from the axioms of set theory.

Version 0.1 of Hmm has been released on October 17th, 2005.

The development version can be found at http://www.solcon.nl/mklooster/repos/hmm/. This is a darcs repository (→ 6.6).

Hmm can't currently do more than just read and verify a Metamath file. However, the longer-term goal is to generate calculational proofs from Metamath proofs. As an example, the Metamath proof that cross-product distributes over union (see http://us.metamath.org/mpegif/xpundi.html) could be visualized something like this:

```
   ( ( A X. B ) u. ( A X. C ) )
=      "LHS of u.: (df-xp); RHS of u.: (df-xp)"
   ( { <. x, y >. | ( x e. A /\ y e. B ) }
     u. { <. x, y >. | ( x e. A /\ y e. C ) } )
=      "(unopab)"
   { <. x, y >. | ( ( x e. A /\ y e. B )
                  \/ ( x e. A /\ y e. C ) ) }
=      "in pair comprehension: (andi)"
   { <. x, y >. | ( x e. A
                  /\ ( y e. B \/ y e. C ) ) ) }
=     "in pair comprehension: RHS of /\: (elun)"
   { <. x, y >. | ( x e. A
                  /\ y e. ( B u. C ) ) }
=      "(df-xp)"
   ( A X. ( B u. C ) )
```

This proof format would make it easier to understand Metamath proofs.

I am working towards this goal, slowly and step by step.

Further reading

http://www.solcon.nl/mklooster/repos/hmm/

4.2.5 GSLHaskell

Report by:	Alberto Ruiz
Status:	active development

GSLHaskell is a high level functional interface to some linear algebra computations and other numerical routines, internally implemented using the GNU Scientific Library. The goal is to achieve the functionality and performance of GNU-Octave or similar systems.

The library is in a preliminary status, but the binding infrastructure is nearly finished and some simple applications requiring basic linear algebra (real svd and qr factorizations, symmetric eigensystems, etc.), numeric integration and differentiation, multidimensional minimization, etc., can already be written.

The immediate developments include a testing suite, updating the manual, writing additional illustrative examples, and some code refactoring. Then we can proceed to include the interface for the remaining functions and implement, using additional libraries, some useful algorithms in Octave not currently available in the GSL.

Further reading

http://dis.um.es/~alberto/GSLHaskell

4.2.6 An Index Aware Linear Algebra Library

Report by:	Frederik Eaton
Status:	unstable; actively maintained

The index aware linear algebra library is a Haskell interface to a set of common vector and matrix operations. The interface exposes index types and ranges to the type system so that operand conformability can be statically guaranteed. For instance, an attempt to add or multiply two incompatibly sized matrices is a static error. A prepose-style (i.e. following Kiselyov and Chan's "Implicit Configurations" paper) approach is used for generating type-level integers for use in index types. Vectors can be embedded in a program using a set of template Haskell routines.

Currently the library is in a "proof-of-concept" state. The interface has an example implementation using Arrays, but ultimately it should be primarily used with a fast external linear algebra package such as ATLAS. I would like to see it become part of Alberto Ruiz's GSL library (→ 4.2.5), which can be used with ATLAS, and he has expressed an interest in adopting it. That is why I haven't given it a real name yet.

The original announcement is here:

Further reading

- Original announcement: http://article.gmane.org/gmane.comp.lang.haskell.general/13561
- Library: http://ofb.net/~frederik/futility/src/Vector/Base.hs http://ofb.net/~frederik/futility/src/Vector/Array.hs http://ofb.net/~frederik/futility/src/Vector/Template.hs http://ofb.net/~frederik/futility/src/Domain.hs http://ofb.net/~frederik/futility/src/Prepose.hs http://ofb.net/~frederik/futility/src/Vector/read-example.hs http://ofb.net/~frederik/futility/src/Vector/examples.hs

4.2.7 Ivor

Report by:	Edwin Brady
Status:	active development

Ivor is a tactic-based theorem proving engine with a Haskell API. Unlike other systems such as Coq and Agda, the tactic engine is primarily intended to be used by programs, rather than a human operator. To this end, the API provides a collection of primitive tactics and combinators for building new tactics. This allows easy construction of domain specific tactics, while keeping the core type theory small and independently checkable.

The primary aim of the library is to support research into generative programming and resource bounded computation in Hume (http://www.hume-lang.org/). In this setting, we have developed a dependently typed framework for representing program execution cost, and used the Ivor library to implement domain specific tactics for constructing programs within this framework. However the library is more widely applicable, some potential uses being:

- A core language for a richly typed functional language.
- The underlying implementation for a theorem prover (see first order logic theorem prover example at http://www.dcs.st-and.ac.uk/~eb/Ivor).
- An implementation framework for a domain specific language requiring strong correctness properties.

Ivor features a dependent type theory similar to Luo's ECC with definitions, with additional (experimental) multi-stage programming support. Optionally, it can be extended with heterogenous equality, primitive types and operations, new parser rules and user defined tactics. By default, all programs in the type theory terminate, but in the spirit of flexibility, the library can be configured to allow general recursion.

The library is in active development, although at an early stage. Future plans include development of more basic tactics (for basic properties such as injectivity and disjointness of constructors, and elimination with a motive), a compiler (with optimisations) and a larger collection of standard definitions.

Further reading

http://www.dcs.st-and.ac.uk/~eb/Ivor

4.2.8 magic-haskell

Report by:	John Goerzen
Status:	active development

magic-haskell is a binding to the libmagic library. With magic-haskell, you can determine the type of a file by looking at its contents rather than its name. This library also can yield the MIME type of a file by looking at its contents.

This is often a more useful method than looking at a file's name since it can yield correct results even if a file's extension is missing or misleading.

Further reading

http://quux.org/devel/magic-haskell

4.2.9 MissingH

Report by:	John Goerzen
Status:	active development

MissingH is a library designed to provide the little "missing" features that people often need and end up implementing on their own. Its focus is on list, string, and IO features, but extends into other areas as well. The library is 100% pure Haskell code and has no dependencies on anything other than the standard libraries distributed with current versions of GHC and Hugs.

In addition to the smaller utility functions, recent versions of MissingH have added a complete FTP client and server system, a virtualized I/O infrastructure similar to Python's file-like objects, a virtualized filesys-

tem infrastructure, a MIME type guesser, a configuration file parser, GZip decompression support in pure Haskell, a DBM-style database virtualization layer, and a modular logging infrastructure, complete with support for Syslog.

Future plans for MissingH include adding more network client and server libraries, support for a generalized URL downloading scheme that will work across all these client libraries, and enhancing the logging system.

This library is licensed under the GNU GPL.

Further reading

http://quux.org/devel/missingh

4.2.10 MissingPy

Report by:	John Goerzen
Status:	active development

MissingPy is really two libraries in one. At its lowest level, MissingPy is a library designed to make it easy to call into Python from Haskell. It provides full support for interpreting arbitrary Python code, interfacing with a good part of the Python/C API, and handling Python objects. It also provides tools for converting between Python objects and their Haskell equivalents. Memory management is handled for you, and Python exceptions get mapped to Haskell Dynamic exceptions.

At a higher level, MissingPy contains Haskell interfaces to some Python modules. These interfaces include support for the Python GZip and BZip2 modules (provided using the HVIO abstraction from MissingH), and support for Python DBM libraries (provided using AnyDBM from MissingH (→ 4.2.9)). These high-level interfaces look and feel just like any pure Haskell interface.

Future plans for MissingPy include an expansion of the higher-level interface to include such things as Python regexp libraries, SSL support, and LDAP support.

This library is licensed under the GNU GPL.

Further reading

http://quux.org/devel/missingpy

4.3 Parsing and transforming

4.3.1 Utrecht Parsing Library and Attribute Grammar System

Report by:	Doaitse Swierstra
Status:	Released as cabal packages

The Utrecht parsing Library and the associated Attribute Grammar System have been made available as cabal packages (→ 4.1.1), and as such may be easier to install.

The systems have been succesfully used by Niels van der Velde, one of our Master students, as part of a toolchain to assist in the parallelisation of C code. It seems that the lazy evaluation used inside is requiring quite some memory footprint.

One of our other master students, Joost Verhoog, is about to complete the alternative path to code-generation for the AG system, in which we follow te more traditional multi-pass attribute grammar evaluation schemes, as explained in the thesis of Joao Saraiva http://www.cs.uu.nl/wiki/Swierstra/SupervisedTheses. Our hope is that this will alleviate the aforementioned problem.

4.3.2 Strafunski

Report by:	Joost Visser
Status:	active, maintained
Portability:	Hugs, GHC, DrIFT

Strafunski is a Haskell-based bundle for generic programming with functional strategies, that is, generic functions that can traverse into terms of any type while mixing type-specific and uniform behaviour. This style is particularly useful in the implementation of program analyses and transformations.

Strafunski bundles the following components:

- the library StrategyLib for generic traversal and others;
- precompilation support for user datatypes based on DrIFT (→ 3.4);
- the library ATermLib for data exchange;
- the tool Sdf2Haskell (→ 5.2.6) for external parser and pretty-print integration.

The Strafunski-style of generic programming can be seen as a lightweight variant of generic programming (→ 3.4) because no language extension is involved, but generic functionality simply relies on a few overloaded combinators that are derived per datatype. By default, Strafunski relies on DrIFT to derive the appropriate class instances, but a simple switch is offered to rely on the "Scrap your boilerplate" (→ 3.4) model as available in the Data.Generics library.

Strafunski is used in the HaRe project (→ 5.3.3) and in the UMinho Haskell Libraries and Tools to provide analysis and transformation functionality for languages such as XML Schema, Java, VDM, SQL, spreadsheets, and Haskell itself.

Further reading

http://www.cs.vu.nl/Strafunski/

4.4 System

4.4.1 hs-plugins

Report by:	Don Stewart
Status:	active development

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. Version 1.0rc1 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/

4.4.2 ldap-haskell

Report by:	John Goerzen
Status:	active development

ldap-haskell is a Haskell binding to C-based LDAP libraries such as OpenLDAP. With ldap-haskell, you can interrogate an LDAP directory, update its entries, add data to it, etc. ldap-haskell provides an interface to all the most common operations you would need to perform with an LDAP server.

Further reading

`darcs get` http://darcs.complete.org/ldap-haskell

4.4.3 Package "time" (formerly TimeLib)

Report by:	Ashley Yakeley
Status:	active development

The "time" package replaces the current library for handling time. The "main" modules feature representation of UTC and UT1, as well as the proleptic Gregorian calendar, time-zones, and functions for strftime-style formatting. Additional "second-level" modules handle TAI, leap-seconds, Julian, ISO 8601 week, and "year and day" calendars, calculation of Easter, and POSIX time. Modules are organised under `Data.Time` to distinguish from the old `System.Time`.

The source is in the darcs (→ 6.6) repository "time" in the current standard libraries, and is built by the GHC library build process. Largely complete, remaining work includes putting use examples in the documentation, and integrating existing unit tests into the testsuite repository.

Further reading

http://semantic.org/TimeLib/

4.4.4 The libpcap Binding

Report by:	Dominic Steinitz
Participants:	Greg Wright, Dominic Steinitz

In case anyone is interested, I've put a darcsized copy of the code here:

```
darcs get
    http://www.haskell.org/networktools/src/pcap
```

- Install libpcap. I used 0.9.4.
- `autoheader`
- `autoconf`
- `./configure`
- `hsc2hs Pcap.hsc`
- `ghc -o test test.hs --make -lpcap -fglasgow-exts`

All contributions are welcome.

4.4.5 Streams

Report by:	Bulat Ziganshin
Status:	beta, actively developed

Streams is the new I/O library developed to extend existing Haskell's Handle-based I/O features. It includes:

- Hugs (→ 2.2) and GHC (→ 2.1) compatibility
- Lightning speed (up to 100 times faster)
- UTF-8 and other Char encodings for text I/O
- Various stream types (files, memory-mapped files, memory and string buffers, pipes)
- Binary I/O and serialization facilities (see AltBinary lib (→ 4.7.2))
- Support for streams working in IO, ST and other monads

The main idea of the library is its clear class-based design that allows to split all functionality into a set of small maintainable modules, each of which supports one type of streams (file, memory buffer ...) or one feature (locking, buffering, Char encoding ...). The interface of each such module is fully defined by some type class (Stream, MemoryStream, TextStream), so the library can be easily extended by third party modules that implement additional stream types (network sockets, array buffers ...) and features (overlapped I/O ...).

Further reading

- Documentation page:

http://haskell.org/haskellwiki/Library/Streams
- Download:
  http://freearc.narod.ru/Streams.tar.gz

Contact

⟨Bulat.Ziganshin@gmail.com⟩

4.5 Databases and data storage

4.5.1 CoddFish

Report by:	Alexandra Silva and Joost Visser

The CoddFish library provides a strongly typed model of relational databases and operations on them, which allows for static checking of errors and integrity at compile time. Apart from the standard relational database operations, it provides normal form verification and database transformation operations.

The library makes essential use of the HList library (→ 4.6.7), which provides arbitrary-length tuples (or heterogeneous lists), and makes extensive use of type-level programming with multi-parameter type classes.

CoddFish lends itself as a sandbox for the design of typed languages for modeling, programming, and transforming relational databases.

Further reading

The web site of CoddFish (http://wiki.di.uminho.pt/wiki/bin/view/PURe/CoddFish) provides documentation and a stand-alone release.

4.5.2 Takusen

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

Takusen is a library for accessing DBMS's. It is a low-level library like HSQL, in the sense that it is used to issue SQL statements. Takusen's 'unique-selling-point' is a design for processing query results using a left-fold enumerator. For queries the user creates an iteratee function, which is fed rows one-at-a-time from the result-set. We also support processing query results using a cursor interface, if you require finer-grained control. Currently we fully support Oracle, Sqlite, and PostgreSQL.

Since the last report we have implemented a new internal interface for prepared statements, and improved the PostgreSQL back-end. We have also (very recently!) moved from SourceForge to a darcs repo (→ 6.6) at haskell.org, and hope to present a new release soon from this repo.

Plans for the future include:

- support multiple result-sets (from nested cursors or stored procedures)
- cabalise (→ 4.1.1)
- use Ashley Yakeley's new Time library (→ 4.4.3) instead of CalendarTime
- resurrect MS SQL Server backend
- ODBC backend

Further reading

http://darcs.haskell.org/takusen/devel/

4.5.3 HaskellDB

Report by:	Björn Bringert
Status:	active development and maintenance
Portability:	GHC, Hugs, multiple platforms

HaskellDB is a library for accessing databases through Haskell in a type safe and declarative way. It completely hides the underlying implementation and can interface with several popular database engines through either HSQL or wxHaskell. HaskellDB was originally developed by Daan Leijen. This latest incarnation of HaskellDB was produced as part of a student project at Chalmers University of Technology.

The current version supports:

- Completely type safe queries on databases
- Support for MySQL, PostgreSQL, SQLite and ODBC through HSQL
- Support for ODBC through wxHaskell
- Automatic conversion between Haskell types and SQL types
- Support for bounded strings
- Dynamic loading of drivers via hs-plugins (→ 4.4.1)

Future possible developments include:

- Support for more backends (Oracle)
- Support for non-SQL backends
- Driver-specific code generation. This is needed for non-SQL backends, and we have discovered that no SQL databases implement the standard in quite the same way

There hasn't been a new release for a while, but an experimental Cabalized (→ 4.1.1) version is available in the CVS repository. New developers are very welcome to join the project.

Further reading

http://haskelldb.sourceforge.net/

4.6 Data types and data structures

4.6.1 Standard Collection Libraries (formerly Hierarchical Libraries Collections)

Report by:	Jean-Philippe Bernardy
Status:	stable, maintained

Haskell implementations come with modules to handle Maps, Sets, and other common data structures. We call these modules the Standard Collection Libraries. The goal of this project is to improve those.

Beside incremental improvement of the current code (stress testing, ironing bugs out, small improvements of API, ...), a package has been created to gather collection-related code that would not fit in the base package yet. This includes changes that are either potentially de-stabilizing, controversial or otherwise experimental.

This new package features notably:

- New data structures, including AVL-tree based Maps and Sets (thanks to Adrian Hey);
- A class-based framework for collection data-types, equipped with polymorphic testsuite and benchmarks.

I plan to submit this package to intensive testing and review, in order to be able to include parts of it in the standard.

Further reading

http://hackage.haskell.org/trac/ghc/wiki/CollectionLibraries

4.6.2 The revamped monad transformer library

Report by:	Iavor Diatchki
Status:	mostly stable

Monads are very common in Haskell programs and yet every time one needs a monad, it has to be defined from scratch. This is boring, error prone and unnecessary. Many people have their own libraries of monads, and it would be nice to have a common one that can be shared by everyone. Some time ago, Andy Gill wrote the monad transformer library that has been distributed with most Haskell implementations, but he has moved on to other jobs, so the library was left on its own. I wrote a similar library (before I knew of the existence of Andy's library) and so i thought i should combine the two. The "new" monadic library is not really new, it is mostly reorganization and cleaning up of the old library. It has been separated from the "base" library so that it can be updated on its own.

Since the last report, there has been a new major release of the monad library (version 2.0), and a minor update (version 2.0.1).

Users interested in using the library can download it (with documentation) from the library's website.

Further reading

http://www.cse.ogi.edu/~diatchki/monadLib/

4.6.3 Data.ByteString (formerly FPS (fast packed strings))

Report by:	Don Stewart
Status:	active development

Data.ByteString (formerly FPS) 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 succesfully with GHC 6.4 and 6.5, and hugs March 2005.

Many improvements have been made over the last 6 months, including the new Data.ByteString.Lazy module, successfully used on terabyte data quantities. Data.ByteString will be part of the GHC base libraries in the GHC 6.6 release.

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://www.cse.unsw.edu.au/~dons/code/fps

4.6.4 Edison

Report by:	Robert Dockins
Status:	active development

Edison, a library of efficient data structures for Haskell, now has a new maintainer! A major update of the library – version 1.2 – has just been released.

Major changes from Edison version 1.1 (released in 1999), include:

- API Typeclasses updated to use functional dependencies
- Modules rearranged to use the hierarchical module extension
- Documentation move from separate document to inline Haddock comments (→ 5.5.8)

- Source code is now managed in a darcs repository (→ 6.6)
- New Cabal-based build system (→ 4.1.1)
- A full suite of unit tests, which covers the entire Edison API, is now included
- Numerous additions to the Associated Collection API
- Several new data structure implementations

Further reading

- The project home page may be found at: http://www.eecs.tufts.edu/~rdocki01/edison.html
- The main darcs repository (→ 6.6) is located at: http://www.eecs.tufts.edu/~rdocki01/edison/

4.6.5 Numeric prelude

Report by:	Henning Thielemann
Participants:	Dylan Thurston, Henning Thielemann
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.

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

- complex number,
- residue class,
- infinite precision number in an arbitrary positional system (initial support),
- polynomial, power series, Laurent series
- root set of a polynomial,
- numbers equipped with units (dynamic checks only).

Using the revised system requires hiding some of the standard functions provided by Prelude, which is fortunately supported by GHC (→ 2.1). The library has basic Cabal (→ 4.1.1) support.

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. Ideally each data type resides in a separate module. However this leads to mutual recursive dependencies, which cannot be resolved if type classes are mutually recursive.

A still unsolved problem arises for e.g. residue classes, matrix computations, infinite precision numbers and fixed point numbers. 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.

Further reading

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

4.6.6 2-3 Finger Search Trees

Report by:	Ben Franksen
Status:	new library, not yet released

An efficient implementation of ordered sequences, based on (external, node oriented) 2-3 finger search trees as described in a recent paper by Ralf Hinze (see below).

With regard to asymptotic complexity, 2-3 finger search trees seem to be the best known purely functional implementations of ordered sequences, with the following amortized time bounds:

- `member`, `insert`, `delete`, `split`: $O(\log(\min(d, n - d)))$
- `minimum`, `maximum`, `deleteMin`, `deleteMax`: $O(1)$
- `merge`: $O(ns * \log(nl/ns))$

where d is the distance from the smallest element, ns is the size of the shorter, and nl the size of the longer sequence. These bounds remain valid if the sequence is used persistently.

The project started as an exercise to explore the intriguing possibilities of nested data types to statically check data-structural invariants. One of my interests was to find out how much this helps development in practice. The results are nothing less than impressive to me. I am sure I would *never* have been able to produce anything as complicated with such a (relatively) low effort, had not the type system constantly guided me in the right direction.

Meanwhile, I think this could evolve into a generally useful library. A lot of work remains to be done, though: currently the library provides only the basic functionality and I have just started to get into performance measurements. I suspect some optimizations are possible, but haven't yet looked into it very deeply. The code is mostly tested (and specified, thanks to QuickCheck), but hasn't been used in a real application.

The library is not yet released, mainly because (lacking a personal homepage) I don't have a convenient place on the web to host it. However, I plan to release a first alpha version soon.

Further reading

- Ralf Hinze, *Numerical Representations as Higher-Order Nested Datatypes*, Technical Report IAI-TR-98-12, Institut für Informatik III, Universität Bonn, December 1998
  http://www.cs.bonn.edu/~ralf/publications/IAI-TR-98-12.ps.gz

4.6.7 HList – a library for strongly typed heterogeneous collections

Report by:	Oleg Kiselyov
Developers:	Oleg Kiselyov, Ralf Lämmel, Keean Schupke

HList is a comprehensive, general purpose Haskell library for strongly typed heterogeneous collections including extensible records. HList is analogous of the standard list library, providing a host of various construction, look-up, filtering, and iteration primitives. In contrast to the regular list, elements of HList 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 labels. We have also used HList for type-safe database access in Haskell. HList-based Records form the basis of OOHaskell. The HList library relies on common extensions of Haskell 98.

We added more examples of using HList: pattern-matching on Records; defining a composition of functions in a heterogenous list, using hFoldr. We have incorporated checks for Function and Tuple types (similar to `IsFunction`). We added a rule to the Makefile for the precompilation of the HList library.

We are working on Cabalizing (→ 4.1.1) HList, expanding on the work by Einar Karttunen, and converting HList repository to Darcs (→ 6.6).

Further reading

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

4.6.8 ArrayRef

Report by:	Bulat Ziganshin
Status:	beta

Includes:

- Unboxed references in IO and ST monads, that supports all simple datatypes and IORef/STRef-like interface. This replaces widely used "fast unboxed variables" modules.
- Monad-independent interface to boxed and unboxed references, what allows to implement algorithms executable both in IO and ST monads
- Syntax sugar for references, mutable arrays and hash tables (`=:`, `+=`, `-=`, `.=`, `val`, `ref`, `uref`)
- Refactored implementation of Data.Array.* modules. Changes include support for dynamic (resizable) arrays and polymorphic unboxed arrays
  (http://www.haskell.org/pipermail/haskell-cafe/2004-July/006400.html),

Further reading

- Documentation page:
  http://haskell.org/haskellwiki/Library/ArrayRef
- Download:
  http://freearc.narod.ru/ArrayRef.tar.gz

Contact

⟨Bulat.Ziganshin@gmail.com⟩

4.7 Data processing

4.7.1 HsSyck

Report by:	Audrey Tang
Status:	active development

YAML is a straightforward machine parsable data serialization format designed for human readability and interaction with dynamic languages. It is optimized for data serialization, configuration settings, log files, Internet messaging and filtering.

Syck is an extension, written in C, for reading and writing YAML swiftly in popular scripting languages.

It is part of core Ruby, and also has bindings for Perl 5, Python, Lua, Cocoa, and Perl 6.

HsSyck provides Data.Yaml.Syck as an interface to YAML structures, using Data.ByteString (→ 4.6.3) for efficient textual data representation. Additionally, we provide a set of DrIFT rules (→ 3.4) to dump and load arbitrary Haskell data types in the YAML format.

Further reading

- Subversion repository
  http://svn.openfoundry.org/pugs/third-party/HsSyck/

4.7.2 AltBinary

Report by:	Bulat Ziganshin
Status:	beta, actively developed

AltBinary is a part of the Streams library (→ 4.4.5). AltBinary implements binary I/O and serialization facilities. It features:

- Hugs and GHC compatibility
- Lightning speed (3-20 times faster than GHC Binary)
- Classical get/put Binary class interface
- Full backward compatibility with NewBinary lib
- Byte-aligned and bit-aligned, low-endian and big-endian serialization
- Serialization of all widely used types (integral, enums, float, arrays, maps . . .)
- UTF8 encoding for strings/chars
- Ability to use TH to derive Binary instance for any type
- Over 50 custom serialization routines (putWord32LE, putMArrayWith . . .)
- ability to serialize data to any Stream what implements vPutByte/vGetByte operations, including support for monads other than IO

Further reading

- Documentation page:
  http://haskell.org/haskellwiki/Library/AltBinary
- Download:
  http://freearc.narod.ru/Streams.tar.gz

Contact

⟨Bulat.Ziganshin@gmail.com⟩

4.7.3 Compression-2005

Report by:	Bulat Ziganshin
Status:	stable

Features of the Compression-2005 Library:

- easy and uniform access to most competitive compression algorithms as of April'05: LZMA, PPMd and GRZip
- all input/output performed via user-supplied functions (callbacks), so you can compress data in memory, files, pipes, sockets and anything else
- all parameters of compression algorithm are defined with a single string, for example `"lzma:8mb:fast:hc4:fb32"`.

So, the entire compression program can be written as a one-liner:

```
compress "ppmd:10:48mb" (hGetBuf stdin)
  (\buf size  ->
     hPutBuf stdout buf size >> return size)
```

with decompressor program:

```
decompress "ppmd:10:48mb" (hGetBuf stdin)
  (\buf size  ->
     hPutBuf stdout buf size >> return size)
```

You can replace `"ppmd:10:48mb"` with `"lzma:16mb"` or `"grzip"` to get another two compressors – all three will compress faster and better than bzip2.

Of course, the primary purpose of this library is to give you a possibility to use state-of-the-art compression as an integral part of your Haskell programs.

The only change since the last report is that comprehensive documentation has been added to the library.

Further reading

- Documentation:
  http://haskell.org/haskellwiki/Library/Compression
- Download:
  http://http://freearc.narod.ru/CompressionLibrary.tar.gz

Contact

⟨Bulat.Ziganshin@gmail.com⟩

4.7.4 The Haskell Cryptographic Library

Report by:	Dominic Steinitz

The current release (3.0.3) includes a significant rewrite of the ASN.1 code and is now capable of supporting X.509 attribute certificates as well as identity certificates.

This release contains:

- DES
- Blowfish
- AES
- Cipher Block Chaining (CBC)
- PKCS#5 and nulls padding

- SHA-1
- MD5
- RSA
- OAEP-based encryption (Bellare-Rogaway)
- PKCS#1v1.5 signature scheme
- ASN.1
- PKCS#8
- X.509 Identity Certificates
- X.509 Attribute Certificates

All contributions are welcome.

Further reading

http://www.haskell.org/crypto

4.7.5 2LT: Two-Level Transformation

Report by:	Joost Visser
Participants:	Alcino Cunha, José Nuno Oliveira
Status:	new

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.

A formal and type-safe treatment of two-level transformations has been provided using Haskell, relying in particular on generalized abstract data types (GADTs). The treatment involves a strategic rewrite system, with combinators reminiscent of Strafunski (→ 4.3.2) and Scrap-your-boilerplate (→ 3.4) generics, that allows *type-changing* rewrites rather than type-preserving or type-unifying ones.

A release is available as part of the UMinho Haskell Libraries, and as stand-alone release under the name 2LT. The release includes worked out examples of schema evolution and hierarchical-relational mappings.

Efforts are underway to extend two-level transformations to include not only transformations of types and their values, but also transformations of functions that consume and/or produce such values.

Further reading

- Project URL: http://wiki.di.uminho.pt/wiki/bin/view/PURe/2LT
- Paper: Alcino Cunha, José Nuno Oliveira, Joost Visser. *Type-safe Two-level Data Transformation.* Accepted for publication in Formal Methods 2006, Lecture Notes in Computer Science, July 2006, Springer.

4.8 User interfaces

4.8.1 Gtk2Hs

Report by:	Axel Simon
Maintainer:	Duncan Coutts and Axel Simon
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 MacOS X (it currently uses the X11 server but a native port is in progress).

Gtk2Hs features:

- automatic memory management (unlike some other C/C++ GUI libraries, Gtk+ provides proper support for garbage-collected languages)
- Unicode support
- anti-aliased drawing on screen, PDF, PS, etc. using Cairo
- extensive reference documentation
- an implementation of the Paul Hudak's Haskell School of Expressions 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 and xulrunner rendering engines
- an easy-to-use installer for Windows
- packages for Fedora Core (→ 7.4.2), Gentoo (→ 7.4.4), Debian (→ 7.4.1), FreeBSD and ArchLinux

The Gtk2Hs library is actively maintained and developed. We are about to create a new release containing a completely reworked interface for the list and tree widgets where the contents of these are stored in Haskell data structures. By implementing the interface for a list or tree widget other data structures can be used; in particular, it is possible to directly query a data base.

The releases of Gtk2Hs are tested to run on Windows, Linux, MacOS X (PPC), FreeBSD, OpenBSD and Solaris.

Other news since the last HCAR:

- Gtk2Hs is used for a GUI front-end by the Epigram group (→ 3.3.1) at Nottingham
- SVG drawing in Cairo
- OpenGL drawing
- Bugs are now tracked using Trac at http://hackage.haskell.org/trac/gtk2hs/report

We are about to streamline the handling of signals. Once this is done, we plan to create a release and ensure that the API is stable from that point.

Further reading

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

4.8.2 hscurses

Report by:	Stefan Wehr
Status:	stable/beta

hscurses is a Haskell binding to the ncurses library, a library of functions that manage an application's display on character-cell terminals. hscurses also provides some basic widgets implemented on top of the ncurses binding, such as a text input widget and a table widget.

The binding was originally written by John Meacham http://repetae.net/john/. Tuomo Valkonen http://modeemi.fi/~tuomov/ and Don Stewart http://www.cse.unsw.edu.au/~dons improved it and I finally added some basic widgets and packed it up as a standalone library.

The binding itself is stable; however, the widget library is still beta. I plan to improve and extend the widget library in the next time. The build system will use Cabal (→ 4.1.1) once GHC 6.6 is out.

Further reading

http://www.stefanwehr.de/haskell/

4.9 (Multi-)Media

4.9.1 HOpenGL – A Haskell Binding for OpenGL and GLUT

Report by:	Sven Panne
Status:	stable, actively maintained

The goal of this project is to provide a binding for the OpenGL rendering library which utilizes the special features of Haskell, like strong typing, type classes, modules, etc., but is still in the spirit of the official API specification. This enables the easy use of the vast amount of existing literature and rendering techniques for OpenGL while retaining the advantages of Haskell over lower-level languages like C. Portability in spite of the diversity of Haskell systems and OpenGL versions is another goal.

HOpenGL includes the simple GLUT UI, which is good to get you started and for some small to medium-sized projects, but HOpenGL doesn't rival the GUI task force efforts in any way. Smooth interoperation with GUIs like gtk+hs or wxHaskell on the other hand is a goal, see e.g. http://wxhaskell.sourceforge.net/samples.html#opengl

Currently there are two major incarnations of HOpenGL, differing in their distribution mechanisms and APIs: The old one (latest version 1.05 from 09/09/03) is distributed as a separate tar ball and needs GreenCard plus a few language extensions. Apart from small bug fixes, there is no further development for this binding. Active development of the new incarnation happens in the fptools repository, so it is easy to ship GHC, Hugs, and nhc98 with OpenGL/GLUT support. The new binding features:

- Pure Haskell 98 + FFI
- No GreenCard dependency anymore
- Full OpenGL 1.5 support (NURBS currently only partly implemented), OpenGL 2.0 features like GLSL support are currently under development
- A few dozen extensions
- An improved API, centered around OpenGL's notion of state variables
- Extensive hyperlinked online documentation
- Supports freeglut-only features, too

HOpenGL is extensively tested on x86 Linux and Windows, and reportedly runs on Solaris, FreeBSD, OpenBSD (→ 7.4.3), and Mac OS X. Making the OpenGL and GLUT packages available as separate Cabal (→ 4.1.1) packages is planned.

The binding comes with all examples from the Red Book and other sources, and Sven Eric Panitz has written a tutorial using the new API (http://www.tfh-berlin.de/~panitz/hopengl/), so getting started should be rather easy.

Further reading

http://www.haskell.org/HOpenGL/

4.9.2 HOpenAL – A Haskell Binding for OpenAL and ALUT

Report by:	Sven Panne
Status:	semi-stable, actively maintained

The goal of this project is to provide a binding for OpenAL, a cross-platform 3D audio API, appropriate for use with gaming applications and many other types of audio applications. OpenAL itself is modeled after the highly successful OpenGL API, and the Haskell bindings for those libraries share "the same spirit", too.

Just like OpenGL is accompanied by GLUT, HOpenAL includes a binding for ALUT, the OpenAL Utility Toolkit, which makes managing of OpenAL contexts, loading sounds in various formats and creating waveforms very easy.

The OpenAL and ALUT packages are currently part of the hierarchical libraries shipped with the various Haskell implementations and will be available as separate Cabal (→ 4.1.1) packages soon. They cover the

latest specification releases, i.e. OpenAL 1.1 (EFX extension is under development) and ALUT 1.1.0, and they work on every platform supporting OpenAL and ALUT (Linux, Windows, Mac OS X, BSDs, ...). They are tested with GHC and Hugs and will probably work with other Haskell systems, too, because they use only H98 + FFI.

Further reading

http://www.openal.org/

4.9.3 hsSDL

Report by:	Lemmih
Status:	stable, maintained

hsSDL contains bindings to libSDL, libSDL_gfx, libSDL_image, libSDL_mixer and libSDL_ttf. The bindings can be installed independently of each other and they all require hsc2hs to be built. Some of the bindings are incomplete or lack proper documentation. If you miss a feature please feel free to mail me (Lemmih) a request at ⟨lemmih@gmail.com⟩.

hsSDL differs from the other Haskell SDL bindings by being more complete and properly Cabalized (→ 4.1.1).

Further reading

- Darcs repository:
  http://darcs.haskell.org/~lemmih/hsSDL/
- libSDL:
  http://www.libsdl.org/

4.9.4 Haskore revision

Report by:	Henning Thielemann and 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 (→ 4.1.1) support.

2. The `Music` data type has been generalized 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. Initial support for the real-time software synthesizer SuperCollider through the Haskell interface.

7. The AutoTrack project has been adapted and included.

8. 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 (→ 6.21).)

9. The test suite is now based on QuickCheck and HUnit.

Future plans

Allow modulation of instruments similar to the controllers in the MIDI system. Generate note sheets, say via Lilypond. Connect to other Haskore related projects.

Further reading

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

4.10 Web and XML programming

4.10.1 CabalFind

Report by:	Dimitry Golubovsky
Status:	experimental

CabalFind is an attempt to create a generalized interface to Internet search engines and provide functionality to postprocess search engines' HTML response to extract the necessary information. Initially it was written to collect information about Cabal (→ 4.1.1) package descriptor files (`.cabal`) available over the Internet by issuing specific queries to search engines such as Google and Yahoo (hence the project name was chosen), but may be used for any kind of automated information search, provided that the search criteria are well defined.

CabalFind uses the Haskell XML Toolbox (→ 4.10.4) to query search engines and parse HTML responses.

Further reading

CabalFind is available as a Cabalized package:

```
darcs get
    http://www.golubovsky.org/repos/cabalfind/
```

The Wiki page at http://haskell.org/hawiki/CabalFind/ contains a brief description of the library internals and an example of its usage.

4.10.2 WebFunctions

Report by:	Robert van Herk
Status:	Released as result of my master's thesis project

Project Overview

WebFunctions is a DSEL for developing websites, implemented in Haskell. WebFunctions is a domain specific embedded language for web authoring, implemented in Haskell. The functionality of the WebFunctions framework was inspired by Apple's WebObjects (http:// www.apple.com/WebObjects). We claim it is easier to use since the Haskell type checker makes a lot of extra checks, that are absent from the Apple framework. Unfortunately we do not yet have all the nice tooling and special editors, but we work on this. Some important features of the WebFunctions system are: loose coupling between model, view and controller code, transparent handling of session and application state, the ability to run the whole web application inside a single process, type safe HTML generation and database interaction and abstracted database interaction. For HTML generation, WASH/HTML (→ 4.10.5) is used. HaskellDB (→ 4.5.3) is used for database interaction. An important difference from some of the other Haskell software in the same field is that a WebFunctions application comes with a built-in web server. Because of this, no CGI is used to handle the requests and the state is persistent at the server. This also means no serialization/deserialization of the state is needed. Furthermore, a database abstraction mechanism is implemented that provides the programmer with concurrency support, caching, and transaction management per session. You can download WebFunctions from http://www.cs.uu.nl/wiki/WebFunctions/Releases.

People

Robert van Herk, for whom the development was his master thesis project Doaitse Swierstra, who supervised Robert. Atze Dijkstra, who is one of our local WebObjects experts.

Further reading

http://www.cs.uu.nl/wiki/WebFunctions/WebHome

4.10.3 HaXml

Report by:	Malcolm Wallace
Status:	stable, maintained

HaXml provides many facilities for using XML from Haskell. The public stable release is 1.13, with support for building via Cabal (→ 4.1.1).

In the unstable development version (currently at 1.15, also available through a darcs repository) we have been experimenting successfully with improvements to the secondary parsing stage, where the generic XML tree is re-parsed into typed Haskell trees. We now get good error messages if the parse fails, and the tools DtdToHaskell and DrIFT (→ 3.4) have been updated to use the new framework. Remaining work, before this branch becomes the stable release, is mainly to unify the internal representations of XML type information, so that a generated DTD is accurate, no matter whether the datatype was originally defined in Haskell or in XML.

Further reading

- http://haskell.org/HaXml
- http://www.cs.york.ac.uk/fp/HaXml-1.14
- http://www.ninebynine.org/Software/HaskellUtils/

4.10.4 Haskell XML Toolbox

Report by:	Uwe Schmidt
Status:	fourth major release (current release: 5.5)

Description

The Haskell XML Toolbox 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 validator based on DTDs and a new more powerful validator for Relax NG schemas.

The Haskell XML Toolbox bases on the ideas of HaXml (→ 4.10.3) and HXML, but introduces a more general approach for processing XML with Haskell. Since release 5.1 there is a new arrow interface similar to the approach taken by HXML. This interface is more flexible than the old filter approach. It is also safer, type checking of combinators becomes possible with the arrow interface.

Features

- validating XML parser
- very liberal HTML parser

- XPath support
- full Unicode support
- support for XML namespaces
- flexible arrow interface with type classes for XML filter
- 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 for filter and arrow interface
- Relax NG schema validator based on the arrows interface
- A HXT Cookbook for using the toolbox and the arrow interface

Current Work

Currently a master student works on a project developing a dynamic web application server with servlet functionality. XML and HXT with arrows will be used for all internal data. This project will be finished and the results will be available in September 2006.

In a second master thesis, once again, the development of an XSLT system has been started. We hope to finish this project with more success than with the previous attempts.

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 thesises describing the design of the toolbox, the DTD validator and the arrow based Relax NG validator.

4.10.5 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 cabalized (→ 4.1.1)
- 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 (→ 4.4.1)) 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

- 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.

4.10.6 HAIFA

Report by: Simon Foster

HAIFA is a Web-Service and XML toolkit for Haskell which enables users to both access Web-Service operations as functions in Haskell and publish Haskell functions within Web-Services. The largest single part of HAIFA, is a complex XML serializer library which attempts to make the job of creating de/serializers for Haskell data-types as painless as possible, via the use of both "Scrap Your Boilerplate" lightweight generics and Template Haskell. Our ultimate aim is to make the Web-Service layer transparent with the help of technologies such as XML Schema and WSDL.

HAIFA has been undergoing some substantial work since the last HCAR. Support for extensible hooks has now been dropped, as this makes writing and invoking serializers much simpler and its usefulness was questionable. Extensible hook support was designed primarily as a method of encoding meta-data into a serialization tree using type-classes unknown when the

base serializers were written to bring in meta-data. Due to Haskell's static type-system it is unlikely this could ever have been put to use, and most of its functionality can probably be achieved with value-level generics via meta-data tables.

Apart from this a lot of bugs have been fixed, and HAIFA is now quite useful for doing SOAP services. The library of TH aids for building serializers is also growing, in order to make the job of constructing serializers for complicated data-types as concise as possible. The automatic serializer generator based on SYB is also substantially more intelligent, for example it can now automatically set cardinality constraints for `Maybe` and `[]` typed terms of types automatically.

The newest release also includes some basic XML Schema mapping support, though only from a small subset of Haskell types to XML Schema at the present time. Mapping in the other direction did work before I removed hooks, and once I get round to adapting it, that should work again.

I've also started work on adding support for WSDL, and some of this can be seen in the developmental darcs repository. Development is very slow at the moment due to other commitments, and so I encourage anyone who is interested to get involved in the project.

Further reading

For more information please see the HAIFA project page at http://www.dcs.shef.ac.uk/~simonf/HAIFA.html

4.10.7 HaXR – the Haskell XML-RPC library

Report by:	Björn Bringert
Status:	maintained

HaXR is a library for writing XML-RPC client and server applications in Haskell. XML-RPC is a standard for XML encoded remote procedure calls over HTTP. The library is actively maintained and relatively stable. Since the last report, the library has changed its name to HaXR (thanks to Christopher Milton for the name suggestion), moved its homepage and darcs repo (→ 6.6) to haskell.org, and been Cabalized (→ 4.1.1).

Further reading

http://www.haskell.org/haxr/

5 Tools

5.1 Foreign Function Interfacing

5.1.1 HSFFIG

Report by:	Dimitry Golubovsky
Status:	mostly stable

HSFFIG (HaSkell Foreign Function Interface Generator) is a tool to convert a C header file (`.h`) into Haskell code containing FFI import statements for all entities whose declarations are found in the header file.

A C header file is to be passed through the preprocessor (CPP); output of the preprocessor is piped to the HSFFIG standard input, and the standard output of HSFFIG is to be processed by hsc2hs. The resulting Haskell code contains autogenerated FFI import statements for function prototypes found in the header file (and all header files it includes); `#define` statements and enumerations are converted into haskellized definitions of constants (where possible), and for each structure/union, means are provided for read/write access to members, and to determine amount of memory occupied by the structure or union.

Conceptually, Haskell code generated by HSFFIG gives the Haskell compiler which "connects" a foreign library to an application written in Haskell the same "vision" as the C compiler would have if it were "connecting" the same library to an application written in C using the same header files.

Haskell code interfacing with foreign libraries using HSFFIG may look "almost like C", but under the strict control of the Haskell type system: all information about foreign functions' type signatures is collected automatically.

HSFFIG is intended to be used with the Glasgow Haskell Compiler (→ 2.1), and was only tested for such use.

Known analogs are: c2hs (→ 5.1.3), hacanon.

Further reading

- The HSFFIG project home page: http://hsffig.sourceforge.net/
- Tutorial: http://haskell.org/hawiki/HsffigTutorial/
- The Examples page (Haskell code using HSFFIG with detailed comments): http://haskell.org/hawiki/HsffigExamples/

5.1.2 FFI Imports Packaging Utility

Report by:	Dimitry Golubovsky
Status:	pre-release

FFIPKG (FFI Imports Packaging Utility) is a tool to prepare a Haskell package containing FFI imports for building. It accepts locations of C header and foreign library files as command line arguments and produces Haskell source files with FFI declarations, a Makefile, a Cabal (→ 4.1.1) package descriptor file, and a `Setup.hs` file suitable for running the Cabal package setup program. Standard process of building a package with Cabal (e.g. `runghc Setup.hs` ....) is to follow to actually build and register/install the package.

The utility is a recent addition to the HSFFIG (→ 5.1.1) package.

Of the benefits of packaging FFI imports, all information about (possibly multiple) C header files and libraries (their names and locations) used by Haskell applications is kept with package descriptor: it is only name of the package that needs to be remembered.

The utility is built upon the code base of HSFFIG (→ 5.1.1), and acts as a "driver" running the C preprocessor, the equivalent of the HSFFIG program, and the source splitter.

FFIPKG is intended to be used with the Glasgow Haskell Compiler (→ 2.1) (6.4 and higher), and was only tested for such use.

Current Status

Pre-release. The utility is available from darcs repo (→ 6.6) only. The package installs as HSFFIG-1.1. Updated HSFFIG is also available from this package.

Further reading

- Announce of the pre-release (also contains the darcs repo URL, as well as brief installation instructions): http://article.gmane.org/gmane.comp.lang.haskell.general/13262
- Wiki page (informal user's guide): http://www.haskell.org/haskellwiki/FFI_Imports_Packaging_Utility

5.1.3 C→Haskell

Report by:	Manuel Chakravarty
Status:	active

C→Haskell is an interface generator that simplifies the development of Haskell bindings to C libraries. It reads C header files to automate many tedious aspects of interface generation and to minimise the scope of error in translating C declarations to Haskell.

The last half year saw only minor changes to C→Haskell, but the tool is already sufficiently mature to be used in rather large projects, such as Gtk2Hs (→ 4.8.1). The darcs repository (→ 6.6) of C→Haskell is scheduled to move to http://darcs.haskell.org/ soon with Duncan Coutts becoming a second maintainer. Source and binary packages as well as a reference manual are available from http://www.cse.unsw.edu.au/~chak/haskell/c2hs/.

5.2 Scanning, Parsing, Analysis

5.2.1 Frown

Report by:	Ralf Hinze
Status:	beta, maintained

Frown is an LALR(k) parser generator for Haskell 98 written in Haskell 98.

Its salient features are:

- The generated parsers are time and space efficient. On the downside, the parsers are quite large.
- Frown generates four different types of parsers. As a common characteristic, the parsers are *genuinely functional* (i.e. 'table-free'); the states of the underlying LR automaton are encoded as mutually recursive functions. Three output formats use a typed stack representation, one format due to Ross Paterson (`code=stackless`) works even without a stack.
- Encoding states as functions means that each state can be treated individually as opposed to a table driven-approach, which necessitates a uniform treatment of states. For instance, look-ahead is only used when necessary to resolve conflicts.
- Frown comes with debugging and tracing facilities; the standard output format due to Doaitse Swierstra (`code=standard`) may be useful for teaching LR parsing.
- Common grammatical patterns such as repetition of symbols can be captured using *rule schemata*. There are several predefined rule schemata.
- Terminal symbols are arbitrary variable-free Haskell patterns or guards. Both terminal and nonterminal symbols may have an arbitrary number of synthesized attributes.
- Frown comes with extensive documentation; several example grammars are included.

Furthermore, Frown supports the use of monadic lexers, monadic semantic actions, precedences and associativity, the generation of backtracking parsers, multiple start symbols, error reporting and a weak form of error correction.

Further reading

http://www.informatik.uni-bonn.de/~ralf/frown/

5.2.2 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) to build a complete parser.

Recent changes:

- Alex is now in a Darcs repository (→ 6.6), here: http://cvs.haskell.org/darcs/alex.
- Happy has a new build system, based on Cabal. If you have GHC 6.4.2 (or Cabal 1.1.4 or later), then you should be able to build and install Alex on any platform without requiring any build tools apart from GHC itself.

Further reading

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

5.2.3 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 1.15, released 14 January 2005.

Since that release, the following changes have happened:

- Happy is now in a Darcs repository (→ 6.6), here: http://darcs.haskell.org/happy.

- Happy has a new build system, based on Cabal. If you have GHC 6.4.2 (or Cabal 1.1.4 or later), then you should be able to build and install Happy on any platform without requiring any build tools apart from GHC itself.

- There are some new minor features: the `error` directive lets you define your own error-handling function, and some new production forms to let you get hold of the current token when parsing.

I plan to make a new release in the near future.

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/.

5.2.4 Attribute Grammar Support for Happy

Report by:	Robert Dockins
Status:	active development

I have hacked up Happy (→ 5.2.3) to support attribute grammars. Attribute grammars are a way of annotating context-free grammars to support syntax directed translation and the checking of context-sensitive properties.

What we have:

- Support for attribute grammars using a slight modification to the Happy grammar syntax.
- Haskell 98! No language extensions required.
- Support for all well-defined attribute grammars (conjecture, but I'm pretty sure).

What we don't have:

- Support for GLR parsing (mostly because I don't completely understand it).
- Checks for proper attribute usage.

Simon Marlow, the Happy maintainer, has expressed interest in the extension. I have submitted a patch to him for review and am awaiting a response. In the meantime the darcs patch may be downloaded from http://www.eecs.tufts.edu/~rdocki01/Happy-AttrGrammar.patch

5.2.5 BNF Converter

Report by:	Markus Forsberg
Contributors:	Björn Bringert, Paul Callaghan, Markus Forsberg, Peter Gammie, Patrik Jansson, Antti-Juhani Kaijanaho, Michael Pellauer, and Aarne Ranta
Status:	active

The project started in 2002 as an experiment with Grammatical Framework (GF) where we investigated to what extent GF could be used to generate a compiler front-end for Haskell, i.e. to generate modules such as a lexer, an abstract syntax, and a parser from a GF grammar. This was indeed possible, but we soon realized that some extra special-purpose notation was needed to avoid problems such as reflecting precedence levels in the abstract syntax. To avoid cluttering GF with this special-purpose notation, we wrote a new tool, and hence, the BNF Converter (BNFC) tool was born.

The tool has been actively developed since 2002 and has undergone major development. It is now a multilingual compiler tool. BNFC accepts as input an LBNF (Labelled BNF) grammar, a format we have developed, and generates a compiler front-end (an abstract syntax, a lexer, and a parser). Furthermore, it generates a case skeleton usable as the starting point of back-end construction, a pretty printer, a test bench, and a LaTeX document usable as language specification.

The program components can be generated in Haskell, Java 1.4 and 1.5, C, and C++ using standard parser and lexer tools. It also supports XML generation of the abstract syntax, which is usable for the exchange of data between systems. If the systems are implemented in languages supported by BNFC, the communication can be performed more directly through pretty-printing and parsing the message.

Some highlights:

- used as teaching tool on several CS courses at Chalmers.
- used to develop a telecommunications protocol language compiler at Tieto-Enator.
- used to develop the GF v2.0 language.
- package included in Debian Linux distribution (→ 7.4.1).
- mentioned in Datormagazin, 2005-05, one of the biggest computer magazines in Sweden.

Further reading

- http://www.cs.chalmers.se/ markus/BNFC/
- M. Forsberg, A. Ranta. The BNF Converter: A High-Level Tool for Implementing Well-Behaved Programming Languages. NWPT'02 proceedings, Proceedings of the Estonian Academy of Sciences, December 2003, Tallin, Estonia.
- M. Pellauer, M. Forsberg, A. Ranta: BNF Converter: Multilingual Front-End Generation from Labelled BNF Grammars, Technical Report no.2004-

09 in Computing Science at Chalmers University of Technology and Gothenburg University.

- M. Forsberg, A. Ranta. Tool Demonstration: BNF Converter. HW'2004, Proceedings of the ACM SIGPLAN 2004 Haskell Workshop, Snowbird, Utah.

5.2.6 Sdf2Haskell

Report by:	Joost Visser
Status:	maintained, actively used

Sdf2Haskell is a generator that takes an SDF grammar as input and produces support for GLR parsing and customizable pretty-printing. The SDF grammar specifies concrete syntax in a purely declarative fashion. From this grammar, Sdf2Haskell generates a set of Haskell datatypes that define the corresponding abstract syntax. The Scannerless Generalized LR parser (SGLR) and associated tools can be used to produce abstract syntax trees which can be marshalled into corresponding Haskell values.

The functionality of Sdf2Haskell also includes generation of pretty-print support. From the SDF grammar, a set of Haskell functions is generated that defines a pretty-printer that turns abstract syntax trees back into concrete expressions. The pretty-printer is updateable in the sense that its behavior can be modified per-type by supplying appropriate functions.

Further reading

Sdf2Haskell is distributed as part of the Strafunski bundle for generic programming and language processing (→ 4.3.2).

5.2.7 SdfMetz

Report by:	Tiago Miguel Laureano Alves
Status:	stable, maintained

SdfMetz supports grammar engineering by calculating grammar metrics and other analyses. Currently it supports two different grammar formalisms (SDF and DMS) 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 visualization, showing the non-singleton levels of the grammar, or printing the grammar graph in DOT format. The definition of all except the ambiguity 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.

A web-based interface is planned and more metrics will be add. A front-end to other grammar formalism (yacc and antlr) is also planed. The tool was 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.

5.2.8 XsdMetz: metrics for XML Schema

Report by:	Joost Visser
Status:	new

The XsdMetz tool computes structure metrics and usage metrics for XML document schemas written in the XML Schema format. The computed structure metrics include tree impurity, coupling, cohesion, fan in and out, instability, height, width, and (normalized) count of strong componenents (see: Joost Visser, *Structure Metrics for XML Schema*). The computed usage metrics include XSD-agnostic and XSD-aware counts (see: Ralf Lämmel, Stan Kitsis, and Dave Remy, *Analysis of XML Schema Usage*). The graphs constructed by XsdMetz for the computation of structure metrics can be exported to the *dot* format of GraphViz.

XsdMetz is available as part of the UMinho Haskell Libraries and Tools. A stand-alone release is in preparation.

Further reading

http://wiki.di.uminho.pt/wiki/bin/view/PURe/XsdMetz

5.3 Transformations

5.3.1 The Programatica Project

Report by:	Thomas Hallgren

One of the goals of the Programatica Project is to develop tool support for high-assurance programming in Haskell.

The tools we have developed so far are implemented in Haskell, and they have a lot in common with a Haskell compiler front-end. The code has the potential to be reusable in various contexts outside the Programatica project. For example, it has already been used in the Haskell refactoring project at the University of Kent (→ 5.3.3).

We also have a Haskell source code browser, which displays syntax-highlighted source code where the user can click on any identifier to display its type or jump to its definition.

Further reading

- The Programatica Project, overview & papers: http://www.cse.ogi.edu/PacSoft/projects/programatica/
- An Overview of the Programatica Toolset: http://www.cse.ogi.edu/~hallgren/Programatica/HCSS04/
- Executable formal specification of the Haskell 98 Module System: http://www.cse.ogi.edu/~diatchki/hsmod/
- A Lexer for Haskell in Haskell: http://www.cse.ogi.edu/~hallgren/Talks/LHiH/
- More information about the tools, source code, downloads, etc: http://www.cse.ogi.edu/~hallgren/Programatica/

5.3.2 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 is (or has 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, Pascal Sotin (Université du Rennes).

Status

The previous work lead to the following tools:

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

  http://www.dsic.upv.es/~slucas/csr/termination/muterm

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

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

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

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

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

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

All these tools have been written in Haskell (mainly developed using Hugs and GHC) and use popular Haskell libraries like 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, to appear, 2006.

- 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 termination tools M. Alpuente and S. Lucas 7th International Workshop on Termination, WST'04, pages 6–9, Technical Report AIB-2004-07, RWTH Aachen, 2004.

- 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, to appear, 2005.

5.3.3 HaRe – The Haskell Refactorer

Report by:	Huiqing Li, Chris Brown, Claus Reinke and 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 third 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. The version about to be released (at the time of writing) works with GHC 6.4.2.

In order to allow users to extend HaRe themselves, the latest releases of HaRe include an API for users to define their own program transformations, together with Haddock (→ 5.5.8) 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 backward program slicing techniques based on highlighting sub expressions. There have also been some new refactorings added which work on data types: adding a constructor to a data type and converting a data type into a newtype. The immediate aim for the development of HaRe is to support a number of type-based refactorings.

A snapshot of HaRe is available from our web page, as are recent presentations from the group (including LDTA 05, TFP05), 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 will be available soon after her viva in early May 2006.

Further reading

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

5.4 Testing and Debugging

5.4.1 Tracing and Debugging

Report by:	Olaf Chitil

There exist a number of tools with rather different approaches to tracing Haskell programs for the purpose of debugging and program comprehension. There has been little new development in the area within the last year.

Hood and its variant GHood enable the user to observe the values of selected expressions in a program. Both are easy to use, because they are based on a small portable library. A variant of Hood is built into Hugs.

HsDebug is a gdb-like debugger that is only available from a separate branch of GHC in CVS. The Concurrent Haskell Debugger CHD was extended to support an automatic search for deadlocks.

Further reading

- Hood:
  http://www.haskell.org/hood/
  http://cvs.haskell.org/Hugs/pages/users_guide/observe.html
- CHD:
  http://www.informatik.uni-kiel.de/~fhu/chd/

5.4.2 Hat

Report by:	Olaf Chitil and Malcolm Wallace
Status:	several recent additions

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: hat-observe, hat-trail, hat-detect, hat-delta, hat-explore, hat-cover, hat-anim, black-hat, hat-nonterm ... 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 inter-operate and use a similar command syntax.

Hat can be used both with nhc98 (→ 2.3) 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.

Im May 2005, version 2.04 of Hat was released. Since then numerous bugfixes, several new features and pro-

totype viewing tools, in particular the extended algorithmic debugger hat-delta, have been added in CVS.

Further reading

- Colin Runciman (ed.): *Hat Day 2005: work in progress on the Hat tracing system for Haskell*, Tech. Report YCS-2005-395, Dept. of Computer Science, University of York, UK, October 2005.

  http://www.cs.york.ac.uk/ftpdir/reports/YCS-2005-395.pdf

- Tom Shackell and Colin Runciman: *Faster production of redex trails: The Hat G-Machine*. Trends in Functional Programming, TFP '05, Symposium proceedings.

- A Theory of Tracing Pure Functional Programs;

  http://www.cs.kent.ac.uk/~oc/traceTheory.html

- http://www.haskell.org/hat

5.4.3 buddha

Report by:	Bernie Pope
Status:	inactive

Buddha is a declarative debugger for Haskell 98. It is based on program transformation. Each module in the program undergoes a transformation to produce a new module (as Haskell source). The transformed modules are compiled and linked with a library for the interface, and the resulting program is executed. The transformation is crafted such that execution of the transformed program constitutes evaluation of the original (untransformed) program, plus construction of a semantics for that evaluation. The semantics that it produces is a "computation tree" with nodes that correspond to function applications and constants.

Buddha is freely available as source and is licensed under the GPL. There is also a Debian package (→ 7.4.1), as well as ports to Free-BSD, Darwin and Gentoo (→ 7.4.4).

Nothing new has been added to buddha since the last report. A fairly comprehensive re-write is planned for late 2006.

Further reading

http://www.cs.mu.oz.au/~bjpop/buddha/

5.5 Development

5.5.1 hmake

Report by:	Malcolm Wallace
Status:	stable, maintained

Hmake is an intelligent module-compilation management tool for Haskell programs. It interoperates with any compiler – ghc (→ 2.1), hbc, or nhc98 (→ 2.3) – except jhc (which does not compile modules separately anyway). A new public version was recently released: 3.11, which contains bugfixes and a new *runhs* command. Occasional maintenance and bugfixes continue to the CVS tree at haskell.org.

Further reading

http://haskell.org/hmake/

5.5.2 Zeroth

Report by:	Lemmih
Status:	usable, unmaintained

A program using Template Haskell must link with the TH library even if it contains no references to TH after it has been compiled. Zeroth is a preprocessor which allows modules to use TH without linking with the TH library. To do this, Zeroth evaluates the top level splices from a module and saves the resulting code.

Further reading

- Darcs repository:
  http://darcs.haskell.org/~lemmih/zerothHead/

5.5.3 Ruler

Report by:	Atze Dijkstra
Participants:	Atze Dijkstra, Doaitse Swierstra
Status:	active development

The purpose of the Ruler system is to describe type rules in such a way that a partial Attribute Grammar implementation, and a pretty printed LaTeX can be generated from a description of type rules. The system (currently) is part of the EHC (Essential Haskell compiler) project (→ 3.3.5) and described in a technical paper, which is also included in the PhD thesis describing the EHC project. The system is used to describe the type rules of EHC. The main objectives of the system are:

- To keep the implementation and LaTeX rendering of type rules consistent.

- To allow an incremental specification (necessary for the stepwise description employed by EHC).

Using the Ruler language (of the Ruler system) one can specify the structure of judgements, called judgement schemes. These schemes are used to 'type check' judgements used in type rules and generate the implementation for type rules. A minimal example, where the details required for generation of an implementation are omitted, is the following:

```
scheme expr =
  holes [ | e: Expr, gam: Gam, ty: Ty | ]
  judgespec gam :- e : ty

ruleset expr scheme expr =
  rule app =
    judge A : expr = gam :- a : ty.a
    judge F : expr = gam :- f : (ty.a -> ty)
    -
    judge R : expr = gam :- (f a) : ty
```

This example introduces a judgement scheme for the specification of type rules for expressions, and a type rule for applications (as usually defined in λ-calculus).

Further reading

- Homepage (Ruler is part of EHC): http://www.cs.uu.nl/groups/ST/Ehc/WebHome
  From here the mentioned documentation can be downloaded.

5.5.4 cpphs

Report by:	Malcolm Wallace
Status:	stable, maintained

Cpphs is a robust Haskell replacement for the C preprocessor. 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 don't 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.)

Current release is now 1.2, with two new features: an option to unliterate `.lhs` files during preprocessing, and the ability to install cpphs as a library to call from your own code, in addition to the stand-alone utility.

Further reading

http://haskell.org/cpphs

5.5.5 Visual Haskell

Report by:	Simon Marlow and Krasimir Angelov
Status:	in development

Visual Haskell is a plugin for Microsoft's Visual Studio development environment to support development of Haskell code. It is tightly integrated with GHC, which provides support for intelligent editing features, and Cabal, which provides support for building and packaging multi-module programs and libraries.

The first release of Visual Haskell, version 0.0, was announced on 20 September 2005. It can be obtained from the main Visual Haskell page, here: http://www.haskell.org/visualhaskell/. In order to use Visual Haskell, you need an x86 machine running Windows, and Visual Studio .NET 2003.

Following a relaxation in the license under which Microsoft's Visual Studio SDK is released, we are now able to distribute the source to the plugin under a BSD-style license. The sources are in a darcs (→ 6.6) repository here: http://darcs.haskell.org/vshaskell/. Why not take a look and see what lengths you have to go to in order to write Haskell code that plugs into Visual Studio!

Unfortunately there hasn't been any progress on this project since the release, as the developers have been busy with other projects. We urgently need to update the codebase to work with the latest GHC and make a release that works with Visual Studio 2005. We would be most grateful for any help.

Help is (still) welcome! Please drop us a note: ⟨simonmar@microsoft.com⟩ and ⟨kr.angelov@gmail.com⟩.

5.5.6 hIDE – the Haskell Integrated Development Environment

Report by:	Lemmih
Status:	in the process

Through the dark ages many a programmer has longed for the ultimate tool. In response to this most unnerving craving, of which we ourselves have had maybe more than our fair share, the dynamic trio of #Haskellaniacs (dons, dcoutts and Lemmih) (→ 1.3) hereby announce, to the relief of the community, that a fetus has been conceived:

hIDE – the Haskell Integrated Development Environment.

So far the unborn integrates source code recognition and a chameleon editor, presenting these in a snappy gtk2 environment. Although no seer has yet predicted the date of birth of our hIDEous creature, we hope that the mere knowledge of its existence will spread peace of mind throughout the community as oil on troubled waters. More news will be dispersed as it arises.

5.5.7 Haskell support for the Eclipse IDE

Report by:	Leif Frenzel
Status:	working, though 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 compilers, interpreters, debuggers, documentation generators and other Haskell development tools. Long-term goals include a language model with support for language-aware IDE features, like refactoring and structural search.

The current version is 0.9.1. The project is now maintained by Thiago Arrais.

Every help is very welcome, be it in the form of code contributions, docs or tutorials, or just any feedback if you use the IDE. If you want to participate, please subscribe to the development mailing list (see below).

Further reading

- http://eclipse.org
- http://lists.sourceforge.net/lists/listinfo/eclipsefp-develop
- Project homepage: http://eclipsefp.sf.net

5.5.8 Haddock

Report by:	Simon Marlow
Status:	stable, 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, 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 verison 0.7, released August 4 2005. Version 0.7 contained some major improvements to the way Haddock decides where to hyperlink each identifier in the documentation.

Recent changes:

- There is a new feature to add links to source code and wiki pages from each entity in the Haddock documentation.
- Haddock is now in a Darcs repository (→ 6.6), here: http://darcs.haskell.org/haddock.
- Happy has a new build system, based on Cabal. If you have GHC 6.4.2 (or Cabal 1.1.4 or later), then you should be able to build and install Haddock on any platform without requiring any build tools apart from GHC itself.

Further reading

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

5.5.9 Hoogle – Haskell API Search

Report by:	Neil Mitchell
Status:	v3.0, beta release

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 still under active development, since the last HCAR a complete rewrite has been performed to create version 3, and another rewrite is under way for version 4. Hoogle has moved onto haskell.org and the range of libraries searched has expanded massively, now including most of GHC's libraries. Haddock (→ 5.5.8) now supports generation of Hoogle data files. The future plans are to speed up the program, improve support for libraries and fix some remaining bugs.

Hoogle is available as a web interface, a command line tool and a lambdabot (→ 6.11) plugin.

Further reading

http://haskell.org/hoogle

6 Applications

6.1 h4sh

Report by:	Don Stewart
Status:	active development

h4sh provides a set of Haskell List functions as normal unix shell commands. This allows us to use Haskell in shell scripts transparently.

Each program is generated from the function's type. The supported functions include: `(!!) ($) (++) (:) (\\) concat concatMap cycle delete drop dropWhile elemIndices filter foldl foldr group head id init insert intersect intersperse iterate last length map maximum minimum nub repeat reverse show sort tail take takeWhile transpose unfoldr union words zip`.

Higher order functions use runtime evaluation, allowing arbitrary Haskell code to be passed to, e.g. `map` and `filter`.

h4sh has been ported to the new Data.ByteString (→ 4.6.3) api.

Further reading

- Source and documentation can be found at:
  http://www.cse.unsw.edu.au/~dons/h4sh.html
- The source repository is available:
  `darcs get`
  http://www.cse.unsw.edu.au/~dons/code/h4sh

6.2 Fermat's Last Margin

Report by:	Shae Erisson
Status:	early beta

What is it?

A distributed decentralized wiki-based darcs-backed research paper annotation tool called Fermat's Last Margin.

The problem is that I want to read what other people write in the margins of their research papers. The solution is to share annotations in a darcs repository along with urls to the original paper, thus allowing both distributed operation and no redistribution copyright problems.

How does it work?

In short, wget the pdf/ps, throw it into imagemagick, create wiki pages for the resulting page images, and save text annotations into the darcs repo. If your repo is http accessible, anyone can grab your per-page annotations, and you can grab theirs.

Further reading

- Trac page:
  http://thunderbird.ScannedInAvian.org/flm/
- Demonstration:
  http://thunderbird.scannedinavian.com/~shae/cgi-bin/Flippi?view=TestMargin

6.3 Conjure

Report by:	Shae Erisson
Status:	work in progress

Conjure is a project to write a Bittorrent client in Haskell. The motivations are, a more declarative implementation, better handling of large numbers of torrents, but primarily an opportunity to do something fun. Jesper Louis Andersen is the the primary organizer for Conjure.

Further reading

- Darcs (→ 6.6) repository:
  http://j.mongers.org/pub/haskell/darcs/conjure/

6.4 DEMO – Model Checking for Dynamic Epistemic Logic

Report by:	Jan van Eijck
Participants:	Jan van Eijck, Simona Orzan, Ji Ruan
Status:	active development

DEMO is a tool for modelling change in epistemic logic (the logic of knowledge). Among other things, DEMO allows modeling epistemic updates, graphical display of update results, graphical display of action models, formula evaluation in epistemic models, translation of dynamic epistemic formulas to PDL (propositional dynamic logic) formulas.

Development has started in 2004. DEMO is used for modelling epistemic puzzles and for checking simple communication protocols. Project participants are Jan van Eijck, Simona Orzan and Ji Ruan.

Source code and documentation are available from the project web page.

Immediate plans are to extend the tool, to apply it to model checking of more involved communication protocols, and to improve the documentation.

Further reading

http://www.cwi.nl/~jve/demo/

6.5 Pugs

Report by:	Audrey Tang
Status:	active development

Started on February 1st 2005, Pugs is an implementation of the Perl 6 language, including a full-fledged parser and runtime, as well as compiler backends targetting JavaScript, Perl 5 and the Parrot virtual machine. It also supports inline Haskell and Perl 5 code in Perl 6 modules, as well as dynamic Haskell evaluation through the hs-plugins (→ 4.4.1) package.

As of this writing, we are working closely with Larry Wall and other language designer to synchronize the specification with our implementation, so that Pugs can become a fully-conforming self-hosting Perl 6 implementation.

The Pugs team has over 200 committers from Haskell, Perl, Python, Ruby, JavaScript and other language communities; the *Learning Haskell* and *Introduction to Pugs* set of talks, published at the Pugs homepage, were also welcomed in several Open Source conferences. Join us on irc.freenode.net `#perl6` to participate in the development!

Further reading

- Development journal
  http://pugs.blogs.com/
- Pugs homepage
  http://pugscode.org/
- Subversion repository
  http://svn.openfoundry.org/pugs/

6.6 Darcs

Report by:	David Roundy
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 very easy to use tool for every day use because it follows the principle of keeping simple things simple.

Darcs is free software licensed under the GNU GPL.

Further reading

http://darcs.net

6.7 Arch2darcs

Report by:	John Goerzen
Status:	active development

Arch2darcs is a Haskell application designed to help convert tla/Arch repositories to Darcs (→ 6.6) repositories while preserving as much history as practical. Arch2darcs is written in pure Haskell.

Further reading

`darcs get` http://darcs.complete.org/arch2darcs

6.8 downNova

Report by:	Lemmih

'downNova' is a program designed for automating the process of downloading TV series from mininova.org. It will scan your downloaded files to find out what your interests are and download missing/new episodes to your collection. Advanced classification techniques are used to interpret the file names and 'downNova' will correctly extract series name, season number, episode number and episode title in nigh all cases. This might be abused for illegally downloading copyrighted material. That is however not the intended use of this program and I do not condone such activities.

Further reading

- Darcs repository:
  http://darcs.haskell.org/~lemmih/downNova/
- mininova:
  http://www.mininova.org/

6.9 HWSProxyGen

Report by:	André Furtado

HWSProxyGen is a web services proxy generator for the Haskell functional language, implemented in Haskell and C#. The final purpose is to show that Haskell and functional languages in general can be used as a viable way to the implementation of distributed components and applications, interacting with services implemented in different languages and/or platforms.

The first beta version of HWSProxyGen (0.1) was released in March/2005. It is restricted to generating proxies only to web services created with Visual Studio .NET. Other web services *can* work with HWSProxyGen, but this is not assured by this first version, since

they can contain unsupported XML elements in their description.

HWSProxyGen is free. Its binaries and source code are available at the project website: http://www.cin.ufpe.br/~haskell/hwsproxygen. The project was created by the Informatics Centre of Federal University of Pernambuco (UFPE). Extensions and enhancements are welcome.

In the last months, an English version of the HWSProxyGen technical paper was created and is available in the References section of the project website. Although HWSProxyGen is being used experimentally in some academic projects at UFPE, there are no immediate plans for it and future versions are still not planned yet.

Further reading

- Web Services Developer Center
  http://msdn.microsoft.com/webservices/
- Microsoft.NET
  http://www.microsoft.com/net
- World Wide Web Consortium
  http://www.w3.org/
- The Haskell.NET Project
  http://www.cin.ufpe.br/~haskell/haskelldotnet
- Haskell HTTP Module (by Gray W. & Bringert B.) (→ 4.10.7)

6.10 Hircules, an irc client

Report by:	Jens Petersen

Hircules is a gtk2-based IRC client built on gtk2hs (→ 4.8.1) and code from lambdabot (→ 6.11). The last release was version 0.3. I recently updated my tree to build with the current releases of ghc and gtk2hs and I am planning to import it to darcs.haskell.org soon to make it easier for other people to contribute patches.

Further reading

http://haskell.org/hircules/

6.11 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. Version 3.1 of lambdabot has been tagged, and development continues. Lambdabot reached the 10k lines of code mark over the last 6 months.

Further reading

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

6.12 λFeed

Report by:	Manuel Chakravarty
Status:	active

Drive your blog with Haskell! λFeed generates RSS 2.0 feeds and corresponding HTML from a non-XML, human-friendly format for channels and news items. Currently, many desirable features are still missing. However, the internal representation of RSS 2.0 feeds is already rather feature-full; it includes, for example, enclosure as needed for podcasts. More information and the darcs repository is available from http://www.cse.unsw.edu.au/~chak/haskell/lambdaFeed/.

6.13 yi

Report by:	Don Stewart
Status:	maintained

yi is a project to write a Haskell-extensible editor. yi is structured around an basic editor core, such that most components of the editor can be overridden by the user, using configuration files written in Haskell. Version 0.1.0 has been released, and provides vim, vi and nano emulation, through an ncurses interface. yi is stable and maintained, though active development has slowed.

Further reading

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

6.14 Dazzle

Report by:	Martijn Schrage and Arjan van IJzendoorn

Dazzle is a graphical toolbox for Bayesian networks that is developed by the Decision Support System group of Utrecht University. It is written in Haskell

and uses wxHaskell as its GUI library. For inference it uses the C++ library SMILE, developed by the Decision Systems Laboratory of Pittsburgh University. Dazzle's features include browsing cases, test selection, logic sampling and sensitivity analysis. The application runs on both Windows and Linux. The project has produced several spin-offs: a progress indicator for pure algorithms, an abstraction for persistent documents, and the XTC library for typed controls. The Dazzle toolbox itself is closed source, but the spin-off libraries are available from the web page.

Further reading

http://www.cs.uu.nl/dazzle/

6.15 Blobs

Report by:	Malcolm Wallace
Status:	experimental

Blobs is a diagram editor for directed graphs, written in Haskell using the platform-independent GUI toolkit wxHaskell. It is based on the Dazzle (→ 6.14) tool presented at the Haskell Workshop in Tallinn, but omitting the proprietary Bayesian analysis algorithms. Blobs is an open project, designed to be a capable (but fairly generic) drawing and editing front-end, so we can share the main GUI effort amongst several different back-end analysis tools.

We are at a fairly early stage of development – if you need a graph editor, please get involved and help to improve it!

What can Blobs do?

- Draw nodes with textual labels, and optional extra (polymorphic) information labels.
- Connect nodes together with edges. An edge has optional extra information labels.
- You can create palettes of different node shapes, and load a palette into the editor. (Currently, palette creation is by hand, not graphical.)
- Graphs are stored in an XML file format.
- If you have a backend engine, you can send the graph to it for analysis, receiving a graph back for viewing as a result.

Further reading

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

6.16 INblobs – Interaction Nets interpreter

Report by:	Miguel Vilaca
Status:	active, maintained
Portability:	Windows, Linux and partially in Mac (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 (→ 6.15) from Arjan van IJzendoorn, Martijn Schrage and Malcolm Wallace.

It's in the first release but a bundle of new features is ready to be added.

The tool is being developed using the repository system Darcs (→ 6.6).

Further reading

http://haskell.di.uminho.pt/jmvilaca/INblobs/

6.17 Yarrow

Report by:	Frank Rosemeier
Status:	stable

From the Yarrow web pages:

"A proof-assistant is a computer program with which a user can construct completely formal mathematical proofs in some kind of logical system. In contrast to a theorem prover, a proof-assistant cannot find proofs on its own.

"Yarrow is a proof-assistant for Pure Type Systems (PTSs) with several extensions. A PTS is a particular kind of logical system, defined in

> Henk P. Barendregt: Lambda Calculi with Types; in D.M. Gabbai, S. Abramsky, and T.S.E. Maibaum (editors): Handbook of Logic in Computer Science, volume 1, Oxford University Press, 1992.

"In Yarrow you can experiment with various pure type systems, representing different logics and programming languages. A basic knowledge of Pure Type Systems and the Curry-Howard-de Bruijn isomorphism is required. (This isomorphism says how you can interpret types as propositions.) Experience with similar proof-assistants can be useful."

In 2003 Frank Rosemeier has ported Yarrow (written by Jan Zwanenburg using Haskell 1.3, see http://www.cs.kun.nl/~janz/yarrow/) to Haskell 98. Now the Haskell 98 source code is available from his *web page* using the address

http://www.rosemeier.info/frank.prog.en.html.

The new *Yarrow homepage* is located at

http://www.haskell.org/yarrow/.

There you can find a copy of the homepage for the Haskell 1.3 version as well as the Haskell 98 adaption. Please let me know if you have some idea to improve the webpages. From my homepage http://www.rosemeier.info you can send an e-mail to me.

6.18 DoCon, the Algebraic Domain Constructor

Report by: Serge Mechveliani

DoCon is a program for *symbolic computation in mathematics*, written in Haskell (using extensions such as multiparametric classes, overlapping instances, and other minor features). It is a package of modules distributed freely, with the source program and manual.

DoCon, the Algebraic Domain Constructor, version 2.08 has been released in 2005. It is available on the public sites.

Real DoCon development has stopped before 2002. At the moment, only the GHC system-dependent changes are considered. Probably, there will be a new release (version 2.09) which runs under GHC (→ 2.1) 6.4 or later. This is a matter of correcting the GHC system usage in the manual.

Further reading

http://haskell.org/docon/

6.19 Dumatel, a prover based on equational reasoning

Report by: Serge Mechveliani

Dumatel is a *prover based on term rewriting and equational reasoning*, written in Haskell (using extensions such as multiparametric classes, overlapping instances). It is a package of modules distributed freely, with the source program and manual.

Dumatel, a prover based on equational reasoning, version 1.02, has been released in 2005. It is available on the public sites. The current 1.02 program appears to have many bugs. A new, improved version is currently being prepared.

Also available is the copy of a talk given during the Groebner Semester 2006 in Linz, Austria: Serge D. Mechveliani, *A design for predicate calculus prover based on completion*, http://www.ricam.oeaw.ac.at/srs/groeb/program.php.

Further reading

http://haskell.org/dumatel/

6.20 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 customizable by means of formatting directives that are interpreted by lhs2TeX. Other directives allow the selective inclusion of program fragments, so that multiple versions of a program and/or document can be produced from a common source. The input is parsed using a liberal parser that can interpret many languages with a Haskell-like syntax, and does not restrict the user to Haskell 98.

The program is stable and can take on large documents.

Since the last HCAR, version 1.11 has been released, that fixes a number of bugs and introduces some minor new features. The program now comes with a library of frequently used formatting directives. Development continues slowly in the Subversion repository.

I would be interested to present some examples of lhs2TeX formatting capabilities on the homepage, and also to extend the lhs2TeX library of formatting directives. If you have written a document that demonstrates nicely what lhs2TeX can do, or if you have designed clever formatting instructions to trick lhs2TeX into doing things previously deemed impossible, please contact me.

Further reading

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

6.21 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 simple signal synthesis backend for Haskore (→ 4.9.4),
- 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,
- framework 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 framework preserves the functional style of programming.

The library comes with basic Cabal (→ 4.1.1) support and requires the Numeric Prelude framework (→ 4.6.5) of revised numeric type classes.

Future plans

Connect with the HaskellDSP library http://haskelldsp.sourceforge.net/. Hope on faster code generated by Haskell compilers. `:-)` Design a common API to the Haskell synthesizer code, CSound support included in Haskore (→ 4.9.4), and the SuperCollider interface.

Further reading

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

7 Users

7.1 Commercial users

7.1.1 Galois Connections, Inc.

Report by: Andy Adams-Moran

Galois (aka Galois Connections, Inc.) is an employee-owned software development company based in Beaverton, Oregon, U.S.A. Galois began life in late 1999 with the stated purpose of using functional languages to solve industrial problems. These days, we emphasize the problem domains over the techniques, and the theme of the recent Commercial User of Functional Programming Workshop (see http://www.galois.com/cufp/) exemplifies our approach: Functional programming as a *means* not an *end*.

Galois develops software under contract, and every project (bar two) that we have ever done has used Haskell; the two exceptions used SML-NJ and OCaml, respectively. We've delivered tools, written in Haskell, to clients in industry and the U.S. government that are being used heavily. Some diverse examples: Cryptol, a domain-specific language for cryptography (with an interpreter and a compiler, with multiple targets); a GUI debugger for a specialized microprocessor; a specialized, high assurance web server, file store, 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.

So, why do we use Haskell? There are benefits to moving to Java or C# from C++ or C, such as cleaner type systems, cleaner semantics, and better memory management support. But languages like Haskell give you a lot more besides: they're much higher level, so you get more productivity, you can express more complex algorithms, you can program and debug at the "design" level, and you get a lot more help from the type system. These arguments have been made time and again though, and they're also pretty subjective.

For Galois, it's also a big bonus that Haskell is close to its mathematical roots, because our clients care about "high assurance" software. High assurance software development is about giving solid (formal or semi-formal) evidence that your product does what it should do. The more functionality provided, the more difficult this gets. The standard approach has been to cut out functionality to make high assurance development possible. But our clients want high assurance tools and products with very complex functionality. Without Haskell (or some similar language), we wouldn't even be able to attempt to build such tools and products.

At Galois, we're happily able to solve real world problems for real clients without having to give up on using the tools and languages we worked on when we were in the Academic world. In fact, we credit most of our success with the fact that we can apply language design and semantics techniques to our clients' problems. Functional languages are an integral part that approach, and a big part of the unique value that our clients have come to known us for.

The good news is that our business is working quite well. As of Fall 2005, Galois is 18 engineers strong, with a support staff of 8. We've been profitable and experienced solid growth each of the last three years.

This year, we've stepped up our community involvement: cvs.haskell.org has moved to a new, much beefier machine that will be funded and maintained by Galois. We're supporting various community efforts on that machine, such as the Hackage database. And we're going to be heavily involved in efforts to codify a new standard Haskell.

We're also trying to drum up support for an industry-based consortium of companies and individuals that use and rely upon Haskell. The stated purpose of the as yet unformed consortium would be to ensure the long-term viability of Haskell, to provide some back-up to the Simons, and to stimulate the development of industrial-grade tools for Haskell development. If you're reading this and are interested in getting involved, e-mail ⟨moran at galois.com⟩.

Further reading

http://www.galois.com/.

7.1.2 Aetion Technologies LLC

Report by: J. Garrett Morris

Aetion Technologies LLC is a small software developer located in Columbus, Ohio, USA. We develop commercial applications of a variety of artificial intelligence techniques, particularly in the application of model-based inference and simulation techniques to decision support and situational awareness, both generating and evaluating new strategies and monitoring and refining existing ones. We are currently focused on defense, with growing applications in finance, manufacturing, and biotechnology.

Our business model requires that we be able to rapidly prototype new systems as well as develop generic software foundations that we can extend to new markets as they open. We have found that Haskell fits

both of these purposes; the majority of our codebase is written in Haskell and compiled using GHC.

We have been hiring aggressively over the past several months, and hope to begin hiring again shortly. As we continue to expand and need to build software that is of more general interest to the community, we hope to release it under a modified BSD license.

Further reading

http://www.aetion.com/

7.1.3 Linspire

Report by: Clifford Beshers

The OS team at Linspire, Inc. would like to announce that we are standardizing on Haskell as our preferred language for core OS development.

We are redoing a bunch of our infrastructure using Haskell as our common standard language. Our first task is redoing our Debian package builder (aka autobuilder) in Haskell. Other tools such as ISO builders, package dependency checkers are in progress. The goal is to make a really tight simple set of tools that will let developers contribute to Freespire, based on Debian tools whenever possible. Our hardware detector, currently in OCaml, is on the block to be rewritten as well.

There are four of us using Haskell, all CCed on this message. All of us have been using functional languages for quite some time. At Linspire, our choices have been OCaml and Haskell. David Fox wrote the hardware detector in OCaml and is now porting it to Haskell. Jeremy Shaw has been doing various utilities in Haskell for several years. Sean Meiners recently wrote an application for managing his recipe collection and is now hooked. I am porting our CD build procedure from OCaml to Haskell.

We are interested in many other uses of Haskell. The recent discussion about Haskell as a shell interests greatly, for example, as we have all suffered through years of bash code. We'd also like to make some Haskell bindings for Qt and KDE, though at the moment we don't have a good plan to tackle that problem efficiently.

To date, Linspire (formerly Lindows) has focused on polishing Linux for the consumer market. I mentioned Freespire, above. We announced Freespire recently (www.freespire.org). Essentially it is a more open, developer friendly version of Linspire. http://freespire.org/about/vision and http://freespire.org/support/faqs have good overviews. Access through apt, open-source CNR client and many other good things.

I mention Freespire because some of our colleagues were concerned that using Haskell would isolate us from the larger community of developers and make it hard to find new employees skilled in Haskell, should we need to. From our perspective, functional programming makes us more effective and we think that getting even a few people who know Haskell hacking with us is a better combination than lots of Perl and bash.

Also, Linspire is based on Debian (→ 7.4.1). We've talked a little with John Goerzen who announced his MissingH library (→ 4.2.9) here a while back. We've imported it and expect to pass updates back to him as well as any other libraries and tools that he would be interested in including in the Debian archive. Also, it seems there are quite a few other libraries out there which are either not debianized or stale. We are looking into helping the folks on the debian-haskell list with that, if possible, documenting and automating wherever possible.

7.2 Haskell in Education

7.2.1 Functional programming at school

Report by: Walter Gussmann

A lot of computer science courses at universities are based on functional programming languages combined with an imperative language. There are many reasons for this: the programming-style is very clear and there are a lot of modern concepts – polymorphism, pattern matching, guards, algebraic data types. There's only little syntax to learn, Finally, the programming code is reduced to a minimum.

Conditions at school

I started teaching functional programming languages at school about 8 years ago in different courses with pupils at age of 16–19 years. Normally they already know an imperative language like Pascal. A good point to perform a paradigm shift to functional programming is recursion.

During the last years I found that learning recursive data structures (queue, stack, list, tree) with Haskell were ideal for classes. They got a much deeper impression about the principles than in imperative or object oriented languages like Pascal or Java.

Especially in high level courses the use of Haskell paid off. The last course about cryptology and theoretical computer science was dominated by Haskell. We implemented a simple RSA-algorithm (with very weak keys) for encoding and decoding of textfiles and some finite deterministic automata. At the end we were able to implement a parser and interpreter for a Pascal-like very simple programming language (not yet published).

Haskell in tests

Haskell was a component of every test, including the German Abitur. These problems seemed to be easier to solve for the pupils, and in tasks with optional languages about 80% chose Haskell. When asked to explain their choice, most of them said that with Haskell they could concentrate on the root of the matter and simplify the problem through a suitable generalization.

Teamwork with Haskell

Last summer I started with a new advanced class. All pupils already visited a one-year-beginners course but they come from 5 different schools and so they have learned five different imperative languages: Pascal, Modula, Python, Java and Delphi. They already knew something about computer science but they were fixed on their first language.

So it was easy for me to start at a very easy level of functional programming. This time I've been concentrating on recursion and developing some projects based on teamwork. First we discussed the electoral system in Germany (Hare-Niemeyer and d'Hondt). Then we implemented a simple version of this system by composing several functions. After designing the structure of each function (with its signature) we implemented them in groups. And we are proud of the result: the main function resolved the problem immediately.

After this positive experience we now do some more complex works, like building the book-index, described in "Haskell: The Craft of Functional Programming" by S. Thompson. Another project draws some lines in a text-window. The line-algorithm is based on a pure recursion.

This kind of teamwork really motivated the pupils. I was impressed about the very short time it took a group of beginners to do such complex programs. We have do some teamwork with Java - but all the projects was much more difficult for the pupils than with Haskell.

What's new?

A few weeks ago I started with an introduction to databases. In the next weeks I'll do some database implementation with Haskell. For the first time we well implement tables as lists of tuples an some SQL-like functions. It's a possibility to make extended use of high-order-functions.

For a more complex example we will read data from a textfile. We use a very short part of the Mondial database, which is available online (http://www.dbis.informatik.uni-goettingen.de/Mondial). This database ist designed as XML-database but can be used as MySQL-database too. The well-formed character of a XML-file can be checked with Haskell.

What is coming in the future?

So there's no question about that: Functional languages are suitable for school. I'm sure that over the years there will be more and more teaching materials, and other teachers will also be convinced of Haskell. For some years I try to persuade other teachers to introduce functional languages through regular workshops, courses and teaching materials.

Today I'm convinced that pupils can understand basic concepts of computer science more easily if they know functional languages like Haskell. The clarity of the language and the modern concept lead to an incredible increase of learned material. My pupils choose Haskell as their favorite of Pascal, C, Java, Haskell and PHP.

Meanwhile the new framework for computer science (in Berlin) includes the obligatory introduction of a declarative language (functional or logical) for advanced courses.

Further reading

http://www.pns-berlin.de/haskell/

7.3 Research Groups

7.3.1 Foundations of Programming Group at the University of Nottingham

Report by: Liyang Hu *et al.*

The Nottingham FoP group is perhaps unique in the UK in bringing functional programming, type theory and category theory together to tackle fundamental issues in program construction. With a total of 25 people, we have a spectrum of interests:

Automated Reasoning Louise Dennis and Matthew Walton are exploring ways of exploiting automated reasoning techniques for dependently-typed programming languages such as Epigram, with a view to extend its verification capabilities. Interesting possibilities arise from giving the programmer control over the techniques used, as well as allowing the program itself to extend the repertoire of available techniques.

Containers Nottingham is the home of the EPSRC grant on *containers* which is a new model of datatypes. We are currently developing the theory and applications of containers.

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 programming model expressive enough to capture object-oriented design patterns.

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. We are currently designing an alternative dependently-typed intermediate language, influenced by our experiences with Epigram.

Epigram Epigram (→ 3.3.1) is a dependently-typed functional programming language in its second reincarnation, implemented in Haskell. Conor McBride heads development with Thorsten Altenkirch, James Chapman, Peter Morris, Wouter Swierstra, Matthew Walton and Joel Wright working on both practical and theoretical aspects of the language.

Quantum Programming Thorsten Altenkirch, Jonathan Grattage and Alex Green are working on quantum computation with the Haskell-like language QML, which introduces quantum data and control structures while integrating reversible and irreversible quantum computation. Guided by its categorical semantics, QML presents a constructive semantics of irreversible quantum computations. A Haskell implementation compiles QML programs into quantum circuits.

Reasoning Catherine Hope, Liyang HU, Graham Hutton and Joel Wright are working on formal reasoning for program correctness and efficiency, where abstract machines play a central rôle.

Exceptions and interrupts are traditionally viewed as being difficult from a semantic perspective. We relate a minimal high-level and low-level semantics containing exceptions via a provably correct compiler, giving greater confidence in our understanding.

Reasoning about *intensional* properties is complicated by non-explicit evaluation order and higher-order functions, but these are eliminated at the abstract machine level. From an evaluator, we can calculate a machine, instrument this with cost information, and backwards derive a high-level function giving space and time usage.

Atomicity deserves particular attention given recent developments in *software transactional memory*. We are devising a low-level semantics featuring commits and aborts, along with a framework to relate this to a high-level *stop-the-world* view.

Short Cut Fusion Short Cut Fusion is used to improve the efficiency of modular programs. Neil Ghani with Tarmo Uustalu, Patricia Johann and Varmo Vene have been developing its theoretical foundations, with much success in both understanding and application of the technique to previously out-of-reach data types. Excitingly, Short Cut Fusion is derived from the principles of initial algebra semantics which underpin Haskell's treatment of datatypes.

Stream Processing Infinite streams support a natural topology. One can represent continuous (with respect to this topology) stream processing functions by datatypes in which induction is nested within coinduction. Peter Hancock, Neil Ghani and Dirk Pattinson have extended this from streams to final coalgebras for a wide class of *container* functors.

Yampa Yampa is an implementation of *functional reactive programming*, maintained by Henrik Nilsson. Some interesting discussions may be found on the yampa-users mailing list. A motion to reanimate the Yampa code base, incorporating recent GADT-based improvements has been submitted to Google's Summer-of-Code. We are hopeful of a new public release soon.

Teaching Haskell plays an important role in the undergraduate programme in Nottingham, via modules in 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 recently completed an introductory Haskell textbook (→ 1.6.2), to be published by Cambridge University Press before the end of 2006.

Events In April, Nottingham successfully played host to the 2006 conferences of the European *Types* project and *Trends in Functional Programming*. Invited speakers included Bart Jacobs, Simon Peyton Jones and Hongwei Xi. This was followed by the *Spring School on Datatype-Generic Programming*, combining theory with practical applications.

The Midlands Graduate School in the Foundations of Computer Science (Easter 2007) will next take place in Nottingham.

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 the FoP group hold an hour-long seminar. We're 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

- Foundations of Programming Group: http://cs.nott.ac.uk/Research/fop/
- Functional Programming at Nottingham: http://sneezy.cs.nott.ac.uk/fp/
- Epigram: http://www.e-pig.org/
- Quantum Programming: http://sneezy.cs.nott.ac.uk/qml/
- Yampa: http://haskell.org/yampa/
- Types 2006: http://cs.nott.ac.uk/types06/
- Trends in Functional Programming 2006: http://cs.nott.ac.uk/~nhn/TFP2006/
- Datatype-Generic Programming 2006: http://cs.nott.ac.uk/ssdgp2006/

7.3.2 Artificial Intelligence and Software Technology at JWG-University Frankfurt

Report by:	David Sabel
Members:	David Sabel, Manfred Schmidt-Schauß

DIAMOND

A current research topic within our DIAMOND project is understanding side effects and Input/Output in lazy functional programming languages using non-deterministic constructs.

We introduced the *FUNDIO* calculus which proposes a non-standard way to combine lazy functional languages with I/O. FUNDIO is a lazy functional core language, where the syntax of FUNDIO has `case`, `letrec`, constructors and an IO-interface: its operational semantics is described by small-step reductions. A contextual approximation and equivalence depending on the Input/Output behavior of normal order reduction sequences have been defined and a context lemma has been proved. This enables us to study a semantics and semantic properties of the language. By using the technique of complete sets of reduction diagrams we have shown a considerable set of program transformations to be correct. Several optimizations of evaluation are given, including strictness optimizations and an abstract machine, and shown to be correct w.r.t. contextual equivalence. Thus this calculus has a potential to integrate non-strict functional programming with a non-deterministic approach to Input/Output and also to provide a useful semantics for this combination.

We applied these results to Haskell by using the FUNDIO calculus as semantics for the GHC core language. Based on an extended set of correct program transformations for FUNDIO, we investigated the local program transformations, which are performed in GHC. The result is that most of the transformations are correct w.r.t. FUNDIO, i.e. retain sharing and do not force the execution of IO operations that are not needed. A detailed description of our investigation is available as a technical report from the DIAMOND project page. By turning off the few transformations which are not FUNDIO-correct and those that have not yet been investigated, we have achieved a FUNDIO-compatible modification of GHC which is called *HasFuse*.

HasFuse correctly compiles Haskell programs which make use of `unsafePerformIO` in the common (safe) sense, since the problematic optimizations that are mentioned in the documentation of the `System.IO.Unsafe` module (let floating out, common subexpression elimination, inlining) are turned off or performed more restrictively. But HasFuse also compiles Haskell programs which make use of `unsafePerformIO` in arbitrary contexts. Since the call-by-need semantics of FUNDIO does not prescribe any sequence of the IO operations, the behavior of `unsafePerformIO` is no longer 'unsafe'. I.e. the user does not have to undertake the proof obligation that the timing of an IO operation wrapped by `unsafePerfomIO` does not matter in relation to all the other IO operations of the program. So `unsafePerformIO` may be combined with monadic IO in Haskell, and since all the reductions and transformations are correct w.r.t. to the FUNDIO-semantics, the result is reliable in the sense that IO operations will not astonishingly be duplicated.

Recently, as a final year project *Hermine Reichau* compared implementations of a natural language interpreter based on the semantics of Montague in Haskell using the Glasgow Haskell compiler and HasFuse together with the underlying call-by-name and call-by-need semantics in the presence of erratic non-determinism. A result is that Montague's natural language semantics is more consistent with call-by-value and call-by-need semantics than with call-by-name semantics.

Non-deterministic Call-by-need Lambda Calculi

Mutual Similarity Important topics are to investigate static analyses based on the operational semantics. In order to do this, more inference rules are necessary for equality in call-by-need lambda-calculi, e.g. a definition of behavioural equivalence. *Matthias Mann* has established a soundness (w.r.t. contextual equivalence) proof for mutual similarity in a non-deterministic call-by-need lambda calculus.

Recently we have shown that Mann's approach using an intermediate "approximation" calculus 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.

Current research is aimed towards extensions of these calculi towards Haskell, e.g. to investigate calculi with a recursive let; to apply the method to non-deterministic call-by-need calculi where a must-convergence is part of the definition of the contextual preorder and to adapt the method to typed languages.

Locally bottom-avoiding choice We investigated a call-by-need lambda-calculus with recursive `let`, `seq`, `case`, constructors and a non-deterministic `amb`-operator, which is locally bottom-avoiding. As equational theory we used contextual equivalence based on may- as well as must-convergence.

In contrast to other approaches our syntax as well as semantics does not make use of a heap for sharing expressions. Instead evaluation is defined as rewriting of `let`-expressions.

We have shown that our equational theory takes fairness into account, and that all deterministic reduction rules and additional program transformations keep contextual equivalence, where the combination of a context lemma together with complete sets of commuting and forking diagrams turned out to be successful.

With the developed proof tools we are able to prove correctness of further program transformations. An analysis of non-terminating terms and subterms should be subject to further investigations. By proving correctness of program transformations used in Haskell compilers and switching off incorrect transformations we could derive a correct compiler for Haskell 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. A technical report is available from our website.

Most implementations of strictness analysis use set constants like $\top$ (all expressions) or $\bot$ (expressions that have no weak head normal form). A current result is that the subset relationship of coinductively defined set constants is in DEXPTIME.

Further reading

- Chair for Artificial Intelligence and Software Technology
  http://www.ki.informatik.uni-frankfurt.de
- DIAMOND – Direct-Call I/O Approach modelled using Non-Determinism
  http://www.ki.informatik.uni-frankfurt.de/research/diamond
- HasFuse – Haskell with FUNDIO-based side effects
  http://www.ki.informatik.uni-frankfurt.de/research/diamond/hasfuse

7.3.3 Formal Methods at Bremen University

Report by:	Christoph Lüth and Christian Maeder
Members:	Christoph Lüth, Klaus Lüttich, 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, static analyzers and proof tools for languages from the CASL family, such as CASL itself, HasCASL, CoCASL, CSPCASL and ModalCASL, and additionally Haskell. HasCASL is a language for specification and development of functional programs; Hets also contains a translation from an executable HasCASL subset to Haskell.

We use the Glasgow Haskell Compiler (GHC 6.4), exploiting many of its extensions, in particular concurrency, multiparameter type classes, hierarchical name spaces, functional dependencies, existential and dynamic types, and Template Haskell. Further tools actively used are DriFT (→ 3.4), Haddock (→ 5.5.8), the combinator library Parsec, HaXml (→ 4.10.3) and Programatica (→ 5.3.1).

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 sucessor 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.

Further reading

- Group activities overview:
  http://www.informatik.uni-bremen.de/agbkb/forschung/formal_methods/
- CASL specification language:
  http://www.informatik.uni-bremen.de/cofi
- Heterogeneous tool set:
  http://www.informatik.uni-bremen.de/cofi/hets
- Proof General Kit
  http://proofgeneral.inf.ed.ac.uk/Kit

7.3.4 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 behavior of the Brooklyn College version of GpH (→ 3.2.2) and the SMP facility of the latest release of GHC (→ 2.1).

In the area of applications, we are working 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 implentation 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⟩

7.3.5 Functional Programming at Macquarie University

Report by: Anthony Sloane
Group leaders: Anthony Sloane, Dominic Verity

Within our Programming Language Research Group we are working on a number of projects with a Haskell focus. Since the last report, work has progressed on the following projects:

- We are close to finishing the first version of a port of the yhc (→ 2.4) runtime to Palm OS handhelds (→ 3.1.1).
- Kate Stefanov has generalised her work on off-the-shelf compression technology for bytecode-based programs from Haskell (nhc98) to Java.
- Matt Roberts continues to focus on languages based on Barry Jay's pattern calculus. We are currently working on semantics of a core language and compilation to an abstract machine.

Further reading

Contact us via email to ⟨plrg@ics.mq.edu.au⟩ or find details on many of our projects at http://www.comp.mq.edu.au/plrg/.

7.3.6 Functional Programming at the University of Kent

Report by: Olaf Chitil

We are a group of about a dozen staff and students with shared interests in functional programming. While our work is not limited to Haskell, it provides a major focus and common language for teaching and research.

Our members pursue a variety of Haskell-related projects, many of which are reported in other sections of this report. Keith Hanna is continuing work on Vital (→ 3.1.2), a document-centered programming environment for Haskell, and on Pivotal (→ 3.1.3), a GHC-based implementation of a similar environment. Mark Callanan is working on type-secure visual editing operations for this kind of environment. Axel Simon maintains the gtk2hs binding to the Gtk+ GUI library (→ 4.8.1) in cooperation with Duncan Coutts, Oxford University. Chris Ryder is improving his Metrics and Visualization library Medina. Huiqing Li, Simon Thompson, Chris Brown and Claus Reinke have released further snapshots of HaRe, the Haskell Refactorer (→ 5.3.3) and started to look at refactoring Erlang programs. Thomas Davie, Yong Luo and Olaf Chitil are working together with the York functional programming group on extending and improving the Haskell tracer Hat (→ 5.4.2).

Further reading

- FP group:
  http://www.cs.kent.ac.uk/research/groups/tcs/fp/
- Vital:
  http://www.cs.kent.ac.uk/projects/vital/
- Pivotal:
  http://www.cs.kent.ac.uk/projects/pivotal/
- Gtk2Hs:
  http://www.haskell.org/gtk2hs
- MEDINA:
  http://www.cs.kent.ac.uk/~cr24/medina/
- Refactoring Functional Programs:
  http://www.cs.kent.ac.uk/projects/refactor-fp/
- Hat:
  http://www.haskell.org/hat/

7.3.7 Parallel and Distributed Functional Languages Research Group at Heriot-Watt University

Report by: Phil Trinder
Members: Abyd Al Zain, Lu Fan, Zara Field, Gudmund Grov, Robert Pointon, Greg Michaelson, Phil Trinder, Jan Henry Nyström, Chunxu Liu, Graeme McHale, Xiao Yan Deng

The Parallel and Distributed Functional Languages (PDF) research group is part of the Dependable Systems Group in Computer Science at the School of Mathematics and Computer Science at Heriot-Watt University.

The group investigates the design, implementation and evaluation of high-level programming languages for high-performance, distributed and mobile computation. The group aims to produce notations with powerful yet high-level coordination abstractions, supported by effective implementations that enable the construction of large high-performance, distributed and mobile systems. The notations must have simple semantics and formalisms at an appropriate level of abstraction to facilitate reasoning about the coordination in real distributed/mobile systems i.e. to transform, demonstrate equivalence, or analyze the coordination properties. In summary, the challenge is to bridge the gap between distributed/mobile theories, like the pi and ambient calculi, and practice, like CORBA and the Globus Toolkits.

Languages

The group has designed, implemented, evaluated and used several high performance/distributed functional languages, and continues to do so. High performance languages include Glasgow parallel Haskell (→ 3.2.2) and Parallel ML with skeletons (PMLS). Distributed/mobile languages include Glasgow distributed Haskell (→ 3.2.3), Erlang (http://www.erlang.org/), Hume (http://www-fp.dcs.st-and.ac.uk/hume/), JoCaml, Camelot, Java Voyager and Java Go.

Projects

Current projects include

- High Level Techniques for Distributed Telecommunications Software 2002–06 is an EPSRC project (GR/R88137) to evaluate high-level distributed programming techniques in a realistic telecommunications context.
- EmBounded Project EU IST-510255 2005–8 that performs the automatic prediction of resource bounds for embedded systems using Hume.
- BAe/DTC SEAS Project SEN 002 2005–7 that engineers embedded software for autonomous vehicle control using optical sensing, again using Hume.
- SCIEnce EU FP6 I3 project (026133) 2006-11 to use GpH to provide access to Grid services from Symbolic Computation systems, including GAP and Maple.

Collaborations

Primary industrial collaborators include groups in Microsoft Research Labs (Cambridge), Motorola UK Research labs (Basingstoke), Ericsson, Agilent Technologies (South Queensferry).

Primary academic collaborators include groups in Complutense Madrid, JAIST, LMU Munich, Phillips Universität Marburg, and St Andrews.

Further reading

http://www.macs.hw.ac.uk/~ceeatia/PDF/

7.3.8 Programming Languages & Systems at UNSW

Report by: Manuel Chakravarty

The PLS research group at the University of New South Wales has produced a couple of Haskell tools, including the interface generator C→Haskell (→ 5.1.3), the hsplugins (→ 4.4.1) library for dynamically loaded type-safe plugins, and the dynamic editor Yi (→ 6.13). Recently, we contributed a new high-performance packed string library `Data.ByteString` (→ 4.6.3), the curses-based mp3 player hmp3, and the RSS 2.0 news feed generator λFeed (→ 6.12).

In cooperation with Microsoft Research, Cambridge, we introduced *associated types* for type classes as a functional alternative to functional dependencies, which are in fact relations, despite the name. Our latest contribution to type-level programming is an improved intermediate language for GHC that unifies the implementation of guarded abstract data types, functional dependencies, and associated types, while simultaneously broadening the range of programs that we can translate. This is joint work with the National University of Singapore and Microsoft Research, Cambridge, available from http://www.cse.unsw.edu.au/~chak/papers/SCP06.html and will be the basis for the implementation of associated types in GHC.

Together with GHC HQ, we just started a new project to finally bring nested data parallelism to GHC, with a focus to utilise multi-core CPUs (→ 3.2.1).

Further details on PLS and the above mentioned activities can be found at http://www.cse.unsw.edu.au/~pls/.

7.4 User groups

7.4.1 Debian Users

Report by: Isaac Jones

The Debian Haskell community continues to grow, with both new users and developers appearing. Together with work on Cabal and libraries (→ 4.1.1) we are working towards providing a much improved Haskell development environment, and the number of applications in Debian written in Haskell is also continuing to grow. A summary of the current state can be found on the Haskell Wiki (→ 1.1): http://www.haskell.org/hawiki/DebianUsers.

For developers, we have a prototype policy for packaging tools for Debian: http://urchin.earth.li/~ian/haskell-policy/haskell-policy.html/.

`dh_haskell` is a tool by John Goerzen to help in building Debian packages out of Cabal packages. It is in the `haskell-devscripts` package.

For users and developers, we have also started a mailing list: http://urchin.earth.li/mailman/listinfo/debian-haskell.

In order to provide backports, bleeding edge versions of Haskell tools, and a place for experimentation with packaging ideas, Isaac Jones and Ian Lynagh have started the "Haskell Unsafe" Debian archive (http://haskell-unsafe.alioth.debian.org/haskell-unsafe.html) where a wide variety of packages can be found. This was recently moved to a Debian server.

7.4.2 Fedora Haskell

Report by: Jens Petersen

Fedora Haskell provides packages of certain Haskell projects for Fedora Core in yum repositories. The main news is that hugs98 (→ 2.2) and gtk2hs (→ 4.8.1) have been added to in Fedora Extras (thanks to Gérard Milmeister). Also ghc (→ 2.1) was updated to 6.4.2 and darcs (→ 6.6) to 1.0.7. I hope more Haskell packages submitted and accepted in Extras in the coming period. There is a mailing list ⟨fedora-haskell@haskell.org⟩ for announcements and questions. Contributions are needed, particular in the form of submissions and reviewing of packages for Fedora Extras.

Further reading

http://haskell.org/fedora/

7.4.3 OpenBSD Haskell

Report by: Don Stewart

Haskell support on OpenBSD continues. A page documenting the current status of Haskell on OpenBSD is at http://www.cse.unsw.edu.au/~dons/openbsd.

GHC (→ 2.1) is available for i386 and amd64. nhc98 (→ 2.3) is available for i386 and sparc. Hugs (→ 2.2) is available for the alpha, amd64, hppa, i386, powerpc, sparc and sparc64. A number of other Haskell tools and libraries are also available, including alex (→ 5.2.2), happy (→ 5.2.3), haddock (→ 5.5.8) and darcs (→ 6.6).

Support for the GHC head branch continues.

7.4.4 Haskell in Gentoo Linux

Report by: Andres Löh

Chris Parrot is the most recent addition to the Gentoo Haskell team, and Lennart Kolmodin will soon become the fifth member.

Most recent work has been centered around improving the ebuilds for GHC, offering Haskell ebuilds for more platforms, and stabilizing packages that have been in the `~arch` part of the tree for quite some time.

We internally use a darcs (→ 6.6) overlay to exchange and test new ebuilds, and coordinate development on IRC (`#gentoo-haskell` on freenode).

In the overlay, there are some packages we consider for addition to the main tree, plus other packages such as a `ghc-darcs` live ebuild that are just provided as unofficial extras.

New ebuilds, comments and suggestions are always welcome. If you file bug reports at bugs.gentoo.org, please make sure that you mention "Haskell" in the subject of the report.

7.5 Individuals

7.5.1 Oleg's Mini tutorials and assorted small projects

Report by: Oleg Kiselyov

The collection of various Haskell mini-tutorials and assorted small projects (http://pobox.com/~oleg/ftp/Haskell/) – has received three additions:

Generic Zipper and a Zipper-based file server/OS

Zipper is a construction that lets us replace an item deep in a complex data structure, e.g., a tree or a term, without any mutation. The result will share as much of

its components with the old structure as possible. The old data structure is still available, and so the changes can be instantly rolled back. Zipper lets us handle a tree or any other enumerable data structure as if it were a stream. Zipper is essentially an 'update' and yet pure functional *cursor* into a data structure. Zipper can be viewed as a delimited continuation reified as a data structure.

Our treatment of zipper is quite different from that of Huet (JFP, 1997) and Hinze and Jeuring (JFP 2001). Our zipper is polymorphic over the data structure to traverse, and the zipper creation procedure is generic and does not depend on the data structure at all. Our zipper is a *derivative* of a traversal function rather than that of a data structure itself.

The articles referenced below introduce the generic zipper and discuss the relationship between zippers and (database) transactions of various isolation modes. We show the updating enumerator and the corresponding zipper that maximally preserve sharing and can walk terms with directed loops. We demonstrate that a zipper can convert a (sequential) map to a fold.

As one of the applications, we present a file server/OS that uses zipper to navigate within a term. If the term in question is a finite map whose keys are strings and values are either strings or other finite maps, the zipper-based file system looks almost the same as the Unix file system. Unlike the latter, however, we offer: transactional semantics; undo of any file and directory operation; snapshots; statically guaranteed the strongest, repeatable read, isolation mode for clients; pervasive copy-on-write for files and directories; built-in traversal facility; and just the right behavior for cyclic directory references.

http://pobox.com/~oleg/ftp/Computation/Continuations.html#zipper

http://pobox.com/~oleg/ftp/Computation/Continuations.html#zipper-fs

Simple fair and terminating backtracking Monad Transformer

The article http://pobox.com/~oleg/ftp/Computation/monads.html#fair-bt-stream presents an implementation of `MonadPlus` and of `MonadPlus` transformer. The back-tracking engine is a hybrid between depth-first and breadth-first evaluators. The engine is *complete*: if the solution exists, it shall be found – even when combining multiple infinite streams (ie., infinitely backtracable computations). The `runM` function also offers a way to limit the search space by setting the maximum number of back-tracking steps. The distinguishing feature of the implementation is its surprising simplicity.

Lightweight dependent typing: eliminating array bound check

Haskell98 with higher-ranked types is already powerful enough to express non-trivial *static* guarantees such as safety of array index operations (i.e., the index being in range of the array bounds). Therefore, we can safely use an efficient `unsafeAt` provided by GHC seemingly for that purpose. Our examples involve native Haskell arrays, index computations, and general recursion. The code is efficient; the static assurances cost us no run-time overhead. The example uses only Haskell98 + higher-ranked types. No new type classes are introduced. The safety is based on: Haskell type system, quantified type variables, and a compact general-purpose trusted kernel.

Our most complex example is folding over multiple, *variously-sized* arrays. This is like a fold over an array – generalized to an arbitrary number of arrays, whose lower and upper index bounds may differ. The index ranges of some arrays do not even have to overlap and may be empty. Neither the number of arrays to process nor their index bounds are statically known. And yet we can statically guarantee that whenever our code accesses any array element, the index is certainly within the bounds of that array. Typing this example in a genuinely dependent type system is probably going to be quite challenging.

For contrast, the article http://pobox.com/~oleg/ftp/Haskell/types.html#dependently-typed-append presents an example of a "heavier-weight" dependent-type programming: appending two lists, assuring that the size of the output list is the sum of the sizes of the two input lists. The lists must therefore be described by a (dependent) type that carries the size of the list. Unlike the lightweight approach, we do not resort to a user-supplied trusted kernel: rather, we exclusively rely on the type system to state and guarantee non-trivial properties of terms.

http://pobox.com/~oleg/ftp/Haskell/types.html#branding

7.5.2 Implementation of "How to write a financial contract"

Report by: Alain Crémieux

The aim is to produce a reference implementation of "Composing contracts: an adventure in financial engineering" (http://research.microsoft.com/Users/simonpj/#contracts-icfp), which could be used as a basis for implementing other DSELs. At present the implementation is divided in 5 layers, from "basic" to "optimizing". Now that GADTs are supported in GHC (→ 2.1), it is possible to express a tagless interpreter for the contract language in a very concise way, even if it is still necessary to guide the type-checker with some annotations. So the next step is

to use Omega, where these annotations are not necessary thanks to the possibility of defining named kinds. And to generalize the contract language to some typed lambda-calculus, including staging. With this I can obtain an optimised interpreter, valuating correctly financial options (the result is easy to check w.r.t. financial books).

Code available on demand.

7.5.3 Inductive Programming

Report by: Lloyd Allison

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

I am continuing to use Haskell to examine what are the products (e.g. Mixture-models (unsupervised classification, clustering), segmentation, classification- (decision-) trees (supervised classification), Bayesian/causal networks/models, time-series models, etc.) of machine learning from a programming point of view, that is how do they behave, what can be done to each one, and how can two or more be combined? The primary aim is the getting of understanding, and that could be embodied in a useful Haskell library or prelude for artificial-intelligence / data-mining / inductive-inference / machine-learning / statistical-inference.

A paper (see below) appeared (1/2006) describing a case-study that defines a learner for the structure and the parameters of a Bayesian network over mixed variables (data attributes): discrete, continuous, and even structured variables; the learner was applied to a Search and Rescue data-set on missing people. This data-set has many missing values which gives great scope for bad puns. IP has also been used to analyse ecological data (submitted) and mutation data on a drug-resistant virus, two applications where IP's flexibility is very useful. A JFP paper (see below) describes an early version of the project. Currently there are types and classes for models (various probability distributions), function-models (regressions), time-series (e.g. Markov models), mixture models, and classification trees (plus regression trees and model trees).

Case-studies include mixtures of time-series, Bayesian networks, time-series models and "the" sequence-alignment dynamic-programming algorithm; a spring-clean of the code is overdue.

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

Future plans

'Inductive programming' seems to be the best name suggested so far, by Charles Twardy, for this kind of programming: 'function' is to 'functional programming' as 'statistical model' is to 'inductive programming'?

External factors slowed progress in 2005 but I hope that things are picking up again. I want to develop time-series models further and am looking at template-Haskell, and similar, for dealing with Excel csv-files in a nice way.

Further reading

- L. Allison. A Programming Paradigm for Machine Learning with a Case Study of Bayesian Networks. ACSC, pages 103–111, January 2006. http://crpit.com/confpapers/CRPITV48Allison.pdf
- L. Allison. Models for Machine Learning and Data Mining in Functional Programming. J. Functional Programming, 15(1), pages 15–32, January 2005. http://dx.doi.org/10.1017/S0956796804005301
- Other reading is listed at the URL: http://www.csse.monash.edu.au/~lloyd/tildeFP/II/

7.5.4 Bioinformatics tools

Report by: Ketil Malde

As part of my PhD work, I developed a handful of (GPL-licensed) tools for solving problems that arise in bioinformatics. I currently have a sequence clustering tool, xsact (currently in revision 1.5), which I believe is one of the more feature-rich tools of its kind. There is also a sequence assembly tool (xtract). In addition, there are various smaller tools that are or were useful to me, and that may or may not be, useful to others. Lately, I've also developed a tool for repeat detection in EST data, called RBR. A beta version is available, but it is fairly thoroughly tested, and I hope to put together a real release soon.

Everything is – of course – available as darcs repos (→ 6.6), at
http://www.ii.uib.no/~ketil/bioinformatics/repos.

Further reading

http://www.ii.uib.no/~ketil/bioinformatics

7.5.5 Using Haskell to implement simulations of language acquisition, variation, and change

Report by: W. Garrett Mitchener
Status: experimental, active development

I'm a mathematician, with expertise in dynamical systems and probability. I'm using math to model language acquisition, variation, and change. My current project is about testing various hypotheses put forth by the linguistics community concerning the word order

of English. Old and Middle English had significantly different syntax than Modern English, and the development of English syntax is perhaps the best studied case of language change in the world. My overall plan is to build simulations of various stages of English and test them against manuscript data, such as the Pennsylvania Parsed Corpus of Middle English (PPCME).

Currently, I'm using a Haskell program to simulate a population of individual agents learning simplified languages based on Middle English and Old French. Mathematically, the simulation is a Markov chain with a huge number of states. Future simulations will probably include sophisticated linguistic computations (parsing and sentence generation) for which Haskell seems to be particularly well-suited. I hope to eventually use the parallel features of GHC to run larger simulations on a PVM grid.

I use GHC and Hugs on Fedora Linux. Oddly enough, the fastest machine in the department for running these computations is my laptop. It's a Pentium M at 1.7 GHz with 2 MB of cache, and for this program, it consistently out-performs my desktop, which is a Pentium 4 at 3 GHz with 1 MB of cache. I suspect the cache size makes the biggest difference, but I haven't done enough experiments to say for sure.

I'm also working on a second Haskell project, which is an interpreted language called Crouton. It's based very loosely on Haskell but without the type system and with much more powerful pattern matching. It will allow me to scan files from the PPCME and other corpora in lisp-like formats, find particular constructions, and transform them. Patterns can be as complex as context free grammars, and apply to whole structures as well as strings. I expect it to be a big help in the data collection part of my language modeling.

Further reading

- http://www.math.duke.edu/~wgm
- http://www.crouton.org