Haskell Communities and Activities Report

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

Ninth Edition – November 20, 2005

Andres Löh (ed.)

Lloyd Allison
Tiago Miguel Laureano Alves
Krasimir Angelov
Alistair Bayley
Jean-Philippe Bernardy
Björn Bringert
Niklas Broberg
Paul Callaghan
Manuel Chakravarty
Olaf Chitil
Koen Claessen
Duncan Coutts
Alain Crémieux
Iavor Diatchki
Atze Dijkstra
Robert Dockins
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
Sven Moritz Hallberg
Thomas Hallgren
Keith Hanna
Bastiaan Heeren
Robert van Herk
Ralf Hinze
Anders Höckersten
Paul Hudak
John Hughes
Graham Hutton
Johan Jeuring
Paul Johnson
Isaac Jones
Oleg Kiselyov
Marnix Klooster
Graham Klyne
Daan Leijen
Lemmih
Huiqing Li
Andres Löh
Rita Loogen
Salvador Lucas
Christoph Lüth
Ketil Malde
Christian Maeder
Simon Marlow
Conor McBride
John Meacham
Serge Mechveliani
Neil Mitchell
William Garret Mitchener
Andy Adams-Moran
J. Garrett Morris
Rickard Nilsson
Sven Panne
Ross Paterson
Jens Petersen
John Peterson
Simon Peyton-Jones
Jorge Sousa Pinto
Bernie Pope
Claus Reinke
Frank Rosemeier
David Roundy
David Sabel
Tom Shackell
Uwe Schmidt
Martijn Schrage
Anthony Sloane
Dominic Steinitz
Donald Bruce Stewart
Martin Sulzmann
Doaitse Swierstra
Wouter Swierstra
Autrijus Tang
Henning Thielemann
Peter Thiemann
Simon Thompson
Phil Trinder
Arjan van IJzendoorn
Tuomo Valkonen
Joost Visser
Malcolm Wallace
Stefan Wehr
Joel Wright
Ashley Yakeley
Bulat Ziganshin

Preface

Finally, here is the 9th edition of the Haskell Communities and Activities Report (HCAR), almost three weeks after the submission deadline. This delay is entirely my own fault. In fact, I have to thank the many contributors to this report even more than usually: never before did I have to ask and push so little; several entries (and quite a few new entries) landed in my inbox before or on the deadline. Thank you very much!

As most of you have probably be waiting for the Report a long time already and are eager to get ahead to the actual contents, let me just enumerate a few technical points:

- I am trying to be more strict about the rule that entries that have not been changed twice are removed. If something is removed, it doesn't mean that is does not exist anymore. But on the other hand, I prefer no content over outdated content, so if in doubt, I remove. All of you who would like to see you entry included again, just submit an entry for the next report!
- As in the previous editions, I've marked entries that have not been in the previous report in blue (or gray, if viewed without color). Entries that have had (at least small) updates have a blue header.
- I have removed the "Haskell events" section. I lack time to keep it up-to-date, and I haven't received any entries from others. I would appreciate if someone would be willing to sub-edit the events section for the next report. Just drop me a mail.
- The next report (the tenth!) will appear in May 2006, so please, already mark the *last weeks of April*, because the new entries will be due by then.

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 haskell.org

Report by: John Peterson and Olaf Chitil

haskell.org belongs to the entire Haskell community – we all have a stake in keeping it as useful and up-to-date as possible. Anyone willing to help out at haskell.org should contact the maintainers to get access to this machine. There is plenty of space and processing power for just about anything that people would want to do there.

What can haskell.org do for you?

- advertise your work: whether you're developing a new application, a library, or have written some really good slides for your class you should make sure haskell.org has a pointer to your work.
- hosting: if you don't have a stable site to store your work, just ask and you'll own haskell.org/yourproject.
- mailing lists: we can set up a mailman-based list for you if you need to email your user community.
- sell merchandise: give us some new art for the cafepress store. Publicize your system with a t-shirt.

The biggest problem with haskell.org is that it is difficult to keep the information on the site current. At the moment, we make small changes when asked but don't have time for any big projects. Perhaps the biggest problem is that most parts (except the wiki) cannot be updated interactively by the community. There's no easy way to add a new library or project or group or class to haskell.org without bothering the maintainers. The most successful sites are those in which the community can easily keep the content fresh. We would like to do something similar for haskell.org.

More and more projects are being hosted on haskell.org. Also, the Haskell Workshop now has a permanent homepage http://haskell.org/haskell-workshop/.

Just what can you do for haskell.org? Here are a few ideas:

- make the site more interactive; allow people to add new libraries, links, papers, or whatever without bothering the maintainers; allow people to attach comments to projects or libraries so others can benefit from your experience; help tell everyone which one of the graphics packages or GUIs or whatever is really useful.
- develop a system where the pages for haskell.org live in a version-controlled repository so that we can more easily share out maintenance.
- add searching capability to haskell.org.

Again, everyone is welcome to join in and help.

Further reading

- http://www.haskell.org/
- http://www.haskell.org/mailinglist.html

1.2 `#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!

1.3 The Haskell HaWiki

Report by: Shae Erisson

The Haskell wikiwiki is a freely editable website designed to allow unrestricted collaboration. The address is http://www.haskell.org/hawiki/. Some highlights are:

- http://www.haskell.org/hawiki/CommonHaskellIdioms
- http://www.haskell.org/hawiki/FundamentalConcepts

Feel free to add your own content!

Sadly, the Haskell wikiwiki has changed to login only editing, due to unmanageable amounts of spam. You can create a user account here: http://www.haskell.org/hawiki/UserPreferences.

1.4 Haskell Weekly News

Report by: John Goerzen

Haskell Weekly News is a newsletter summarizing each week's activity in the Haskell community, and primarily the Haskell mailing lists. Each week's issue is published on the general Haskell list as well as on the Haskell

Sequence. To read HWN online, please visit http://sequence.complete.org/hwn.
Submissions from the public are welcomed in HWN. Information on submitting content to HWN is available at http://sequence.complete.org/hwn-contrib.

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 New textbook – Programming in Haskell

Report by:	Graham Hutton

After many years of work, Programming in Haskell is now finished! The date for publication isn't yet known, but is anticipated to be around early to mid 2006. Further details, including a preview of the book and powerpoint lecture slides for each chapter, are available on the web from http://www.cs.nott.ac.uk/~gmh/book.html.

1.6.2 Haskell Tutorial WikiBook

Report by:	Paul Johnson

I became aware of a placeholder page for a Haskell Wiki textbook over at the WikiBooks project. The URL is http://en.wikibooks.org/wiki/Programming:Haskell.

Since this looks like a Good Thing to have I've made a start. Of course there is no way that little old me could write the entire thing, so I'd like to invite others to contribute.

I'm aware of all the other Haskell Tutorials out there, but they are limited by being single-person efforts with no long term maintenance. This is not meant to denigrate the efforts of their authors: producing even a simple tutorial is a lot of work. But Haskell lacks a complete on-line tutorial that can take a programmer from the basics up to advanced concepts like nested monads and arrows. Once you get past the basics you tend to have to depend on library reference pages and the original academic papers to figure things out.

So what is needed is:

- ◦ Space for a team effort
- ◦ Space to evolve with the language and libraries

A Wikibook offers both of these.

Contributions are welcome. This includes edits to the table of contents (which seems to have been written by someone who doesn't know Haskell) and improvements to my existing text (which I'm happy to concede is not exactly brilliant).

Further reading

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

1.6.3 hs-manpage-howto(7hs)

Report by:	Sven Moritz Hallberg
Status:	active development

The hs-manpage-howto(7hs) is a manpage for documenting Haskell modules with roff manpages. I announced it in the November issue and it has been expanded with some small additions and clarifications since then. Most notable are the guidelines for HISTORY sections in the context of ECT (→4.1.2).

So as before, the hs-manpage-howto(7hs) is a rough document far from complete, meant mainly as a reminder and guide for myself. But if you happen to be writing a Haskell manpage yourself, you should still find it useful.

And if you come up with a guideline not covered, please let me know!

Further reading

http://www.scannedinavian.org/~pesco/man/html7/hs-manpage-howto.7hs.html

2 Implementations

2.1 The Glasgow Haskell Compiler

Report by:	Simon Peyton-Jones

The last six months has been largely a consolidation phase for GHC. With one big exception, there are few new features.

- We released GHC 6.4.1, which contains a multitude of bug-fixes for the 6.4 release. We hope that 6.4.1 will be a stable base for some time to come.
- The big new thing is that we now have a working parallel version of GHC, which runs on a multiprocessor. It's described in our Haskell workshop paper *Haskell on a shared memory multiprocessor*, available at http://research.microsoft.com/~simonpj/papers/parallel/.
- We released Visual Haskell (→ 5.5.5).
- Work continues on packaging GHC as a Haskell library, so that you can call GHC from any Haskell program by saying `import GHC`. This will let you typecheck, compile, and execute all of GHC-supported Haskell from your own application, and should make many Haskell tools much easier to write.

  For the current version of the API, see http://cvs.haskell.org/cgi-bin/cvsweb.cgi/fptools/, and look in ghc/compiler/main/GHC.hs. Please take a look and give us feedback.

There is lots more in the works:

- We are planning to use darcs (→ 6.6) instead of CVS for GHC.
- On the type system front, we hope to
  - extend GHC's higher-rank type system to incorporate impredicative types: http://reserach.microsoft.com/~simonpj/papers/boxy/,
  - fix the GADT implementation to work nicely with type classes,
  - Allow you to use data types as kinds, in a manner similar to Tim Sheard's Omega language.
- We are planning to release GHC 6.6 some time in the next six months. This will include the parallel version of GHC.

As ever, we are grateful to the many people who submit polite and well-characterised bug reports. We're even more grateful to folk actually help develop and maintain GHC. The more widely used GHC becomes, the more we rely on you to help solve people's problems, and to maintain and develop the code. We won't be around for ever, so the more people who are involved the better. If you'd like to join in, please let us know.

2.2 Hugs

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

A new major release of Hugs is planned within a few months. As well as the steady trickle of bug fixes, this will include support for the Cabal infrastructure (→ 4.1.1), Unicode support (contributed by Dmitry Golubovsky) and lots of up-to-date libraries.

It should also include a Windows distribution. Thanks to testing by Brian Smith, the interpreter and the full set of libraries now build under MinGW. Neil Mitchell has rewritten the Windows interface (WinHugs), and this will be in the next release. A prerelease is available for testing on Neil's WinHugs page (http://www-users.cs.york.ac.uk/~ndm/projects/winhugs.php).

Now that Hugs uses Cabal to build its libraries, it will soon be possible to split off libraries as separate Cabal packages, making the core Hugs source distribution smaller. Distributors of binary packages can decide whether to include library packages in an omnibus distribution or distribute them separately.

Support for obsolete non-hierarchical libraries (hslibs and Hugs-specific libraries) will disappear soon.

As ever, volunteers are welcome.

2.3 nhc98

Report by:	Malcolm Wallace
Status:	stable, maintained

nhc98 is a small, easy to install, standards-compliant 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.

We are pleased that a student project, starting with nhc98's sources, is aiming for a complete replacement of most of its parts. The working title is Yhc (→ 2.4), and currently it has an entirely new portable bytecode backend and runtime system. This solves many of the trivial but annoying problems with nhc98, such as lack of support for 64-bit machines, lack of good support for Windows builds, the high-memory bug, and so on.

The other parts of the compiler may be replaced too. The most urgent needs are to:

- overhaul the type inference subsystem, towards the goal of implementing multi-parameter classes;
- add pattern-guards to the source language;
- add concurrent threads using a co-operative scheduler;
- implement exceptions.

Volunteers welcome.

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/

2.5 jhc

Report by:	John Meacham
Status:	unstable, actively maintained, volunteers welcome

jhc is a Haskell compiler which aims to produce the most efficient programs possible via whole program analysis and other optimizations.

Some features of jhc are:

- Full support for Haskell 98, The FFI and some extensions (modulo some bugs being worked on and some libraries that need to be filled out).

- Produces 100% portable ISO C. The same C file can compile on machines of different byte order or bit-width without trouble.

- No pre-written runtime. other than 20 lines of boilerplate all code is generated from the Grin intermediate code and subject to all code simplifying and dead code elimination transformations. As a result, jhc currently produces the smallest binaries of any Haskell compiler. (`main = putStrLn "Hello, World!"` compiles to 6,568 bytes vs 177,120 bytes for GHC 6.4)

- First Intermediate language based on Henk, Pure Type Systems and the Lambda cube. This is similar enough to GHCs core that many optimizations may be implemented in the same way.

- Second Intermediate language is based on Boquist's graph reduction language. This allows all unknown jumps to be compiled out leaving standard case statements and function calls as the only form of flow control. Combined with jhc's use of region inference, this means jhc can compile to most any standard imperative architecture/language/virtual machine directly without special support for a stack or tail-calls.

- Novel type class implementation not based on dictionary passing with many attractive properties. This implementation is possible due to the whole-program analysis phase and the use of the lambda-cube rather than System F as the base for the functional intermediate language.

- Intermediate language and back-end suitable for directly compiling any language that can be embedded in the full lambda cube.

- All indirect jumps are transformed away, jhc's final code is very similar to hand-written imperative code, using only branches and static function calls. A simple basic-blocks analysis is enough to transform tail-calls into loops.

- Full transparent support for mutually recursive modules.

Jhc's ideas are mainly taken from promising research papers that have shown strong theoretical results but perhaps have not been extended to work in a full-scale compiler.

Although jhc is still in its infancy and has several issues to work through before it is ready for public consumption, it is being quickly developed and volunteers are welcome.

Discussion about jhc development currently occurs on gale (gale.org) in the category pub.comp.jhc@ofb.net. A simple web client can be used at yammer.net.

2.6 Helium

Report by:	Bastiaan Heeren
Participants:	Arjan van IJzendoorn, Bastiaan Heeren, Daan Leijen
Status:	stable

The purpose of the Helium project is to construct a light-weight compiler for a subset of Haskell that is especially directed to beginning programmers (see *"Helium, for learning Haskell"*, Bastiaan Heeren, Daan Leijen, Arjan van IJzendoorn, Haskell Workshop 2003). We try to give useful feedback for often occurring mistakes. To reach this goal, Helium uses a sophisticated type checker (→ 3.3.4) (see also *"Scripting the type inference process"*, Bastiaan Heeren, Jurriaan Hage and S. Doaitse Swierstra, ICFP 2003).

Helium has a simple graphical user interface that provides online help. We plan to extend this interface to a full fledged learning environment for Haskell. The complete type checker and code generator has been constructed with the attribute grammar (AG) system developed at Utrecht University. One of the aspects of the compiler is that can log errors to a central repository, so we can track the kind of problems students are having, and improve the error messages and hints.

There is now support for type classes, but this has not been officially released yet. A new graphical interpreter is being developed using wxHaskell (→ 4.5.1), which will replace the Java-based interpreter. The Helium compiler has been used successfully four times during the functional programming course at Utrecht University.

Further reading

http://www.cs.uu.nl/research/projects/helium/

3 Language

3.1 Variations of Haskell

3.1.1 Haskell on handheld devices

Report by:	Anthony Sloane
Status:	unreleased

Work on our port of nhc98 (→ 2.3) to Palm OS is continuing at a slower rate than we would like. A few different strategies have been tried, based on our literate reworking of the nhc98 runtime kernel. The current state is that simple programs can be run on modern Palm hardware. Debugging and broadening of access to the Palm OS library functions is continuing. We hope to have an alpha release for others to try out by the end of the southern hemisphere summer break (February).

3.1.2 Vital: Visual Interactive Programming

Report by:	Keith Hanna
Status:	active (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 feelŠ 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 (→ 4.2.12) to allow dynamic compilation and evaluation of Haskell expressions together with the gtk2hs library (→ 4.5.3) 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.

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:	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.

Currently, the project has produced first versions of the Camila Library and the Camila Interpreter, both distributed as part of the UMinho Haskell Libraries and Tools (→ 7.3.9). 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. The interpreter is implemented with the use of hs-plugins (→ 4.2.12).

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 (→ 7.3.9).

3.1.6 Haskell Server Pages (HSP)

Report by:	Niklas Broberg
Status:	experimental
Portability:	currently posix-specific

Haskell Server Pages is an extension of Haskell for the purpose of writing server-side dynamic webpages. It allows programmers to use syntactic XML fragments in Haskell code, and conversely allows embedded Haskell expressions inside XML fragments. Apart from the purely syntactic extensions, HSP also provides a programming model with datatypes, classes and functions that help with many common web programming tasks. Examples include:

- Maintaining user state over transactions using sessions
- Maintaining application state over transactions with different users
- Accessing query string data and environment variables

HSP can also be seen as a framework that other libraries and systems for web programming could use as a backend.

The HSP implementation comes in the form of a server application intended to be used as a plugin to web servers such as Apache. There is also a one-shot evaluator that could be used to run HSP in CGI mode, however some functionality is lost then, in particular application state. Both the server and the one-shot evaluator rely heavily on hs-plugins (→ 4.2.12).

Currently we have no bindings to enable HSP as a plugin to a webserver. The server can be run in standalone mode, but can then only handle .hsp pages (i.e., no images or the like), or the mentioned one-shot evaluator can be used for CGI. The system is highly experimental, and bugs are likely to be frequent. You have been warned.

Further reading

- Webpage and darcs repo at:
  http://www.cs.chalmers.se/~d00nibro/hsp
- My master's thesis details the programming model and implementation of HSP:
  http://www.cs.chalmers.se/~d00nibro/hsp/thesis.pdf

3.1.7 HASP

Report by:	Lemmih
Status:	active

HASP is a fork of Niklas Broberg's Haskell Server Pages (→ 3.1.6). Changes includes:

- support for all GHC extensions
- front-end based on FastCGI instead of its own web server
- minor bug fixes and performance tuning.

HASP will in the future be using the GHC api (→ 2.1) for faster compilations, better error messages and easier debugging.
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://scannedinavan.org/~lemmih/hasp/
- Original HSP:
  http://www.cs.chalmers.se/~d00nibro/hsp/

3.1.8 Haskell Regular Patterns (HaRP)

Report by:	Niklas Broberg
Status:	stable, currently not actively developed
Portability:	relies on pattern guards, so currently ghc only

HaRP is a Haskell extension that extends the normal pattern matching facility with the power of regular expressions. This expressive power is highly useful in a wide range of areas, including text parsing and XML processing. Regular expression patterns in HaRP work over ordinary Haskell lists (`[]`) of arbitrary type. We have implemented HaRP as a pre-processor to ordinary Haskell.

Further reading

- Webpage and darcs repo at:
  http://www.cs.chalmers.se/~d00nibro/harp/

3.2 Non-sequential Programming

3.2.1 GpH – Glasgow Parallel Haskell

Report by:	Phil Trinder
Participants:	Phil Trinder, Abyd Al Zain, Andre Rauber du Bois, Kevin Hammond, Leonid Timochouk, 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

The first two items are linked by a British Council/DAAD project.

- 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. Preliminary results are published in Al Zain A., Trinder P.W., Loidl H.W., Michaelson G.J *Managing Heterogeneity in a Grid Parallel Haskell*, Practical Aspects of High-level Parallel Programming (PAPP 2005), Atlanta, USA (May 2005).

- We are designing a generic parallel runtime environment encompassing both the Eden (→ 3.2.4) and GpH runtime environments

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

- In separate work 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 Universität Marburg.

GpH Applications

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

- We are participating in the SCIEnce EU FP6 I3 project (026133) to use GpH to provide access to Grid services from Symbolic Computation systems, including GAP and Maple.

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⟩

3.2.2 GdH – Glasgow Distributed Haskell

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

GdH supports distributed stateful interactions on multiple locations. It is a conservative extension of both Concurrent Haskell and GpH (→ 3.2.1), 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 (→ 3.2.1).

Applications and Evaluation

- An EPSRC project *High Level Techniques for Distributed Telecommunications Software* (http://www.macs.hw.ac.uk/~dsg/telecoms/, GR/R88137) is now underway and is entering its first GdH phase. The project evaluates GdH and Erlang in a telecommunications context, the work is a collaboration between Heriot-Watt University and Motorola UK Research Labs.
- There is a forthcoming Ph.D. thesis on the design, implementation and use of GdH by Robert Pointon (http://www.macs.hw.ac.uk/~rpointon/).

Further reading

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

3.2.3 Mobile Haskell (mHaskell)

Report by:	Phil Trinder
Participants:	Andre Rauber du Bois, Hans-Wolfgang Loidl, Phil Trinder

Mobile Haskell supports both strong and weak mobility of computations across open networks. The mobility primitives are higher-order polymorphic channels. Mid-level abstractions like remote evaluation, analogous to Java RMI, are readily constructed. High-level mobility skeletons like mobile map and mobile fold encapsulate common patterns of mobile computation.

Status

An alpha-release release of mHaskell is available on request from the mHaskell homepage. A number of applications have been constructed in mHaskell, including a stateless webserver, a distributed meeting planner and a mobile agent system. Mobility skeletons are being implemented in mobile Javas.

Further reading

- The mHaskell homepage http://www.macs.hw.ac.uk/~dubois/mhaskell

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, Rafael Martínez, 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 (→3.2.1). 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.

Recent and Forthcoming Publications

Survey and new 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)

Skeletons

- Jost Berthold, Rita Loogen: *The Impact of Dynamic Channels on Functional Topology Skeletons*, Third International Workshop on High-level Parallel Programming and Applications (HLPP 2005), Warwick University, UK, July 2005.
- Jost Berthold, Rita Loogen: *Skeletons for Recursively Unfolding Process Topologies*, Mini-Symposium "'Algorithmic Skeletons and High-Level Concepts for Parallel Programming"', Conference on Parallel Computing (ParCo 2005), September 2005.
- M. Hidalgo-Herrero, Y. Ortega-Mallén, F. Rubio: *Towards Improving Skeletons in Eden*, Mini-Symposium "'Algorithmic Skeletons and High-Level Concepts for Parallel Programming"', Conference on Parallel Computing (ParCo 2005), September 2005.
- Clara Segura, Ricardo Peña: *Reasoning About Skeletons in Eden*, Mini-Symposium "'Algorithmic Skeletons and High-Level Concepts for Parallel Programming"', Conference on Parallel Computing (ParCo 2005), September 2005.

Compilation and Profiling

- Steffen Priebe: *Preprocessing Eden with Template Haskell*, ACM SIGPLAN Generative Programming and Component Engineering (GPCE '05), Tallinn, Estonia, LNCS 3676 (pp. 357-372), Springer-Verlag 2005.
- Carmen Torrano and Clara Segura: *Strictness Analysis and let-to-case Transformation using Template Haskell*, Symposium on Trends in Functional Programming (TFP 2005), Tallinn, Estonia, September 2005.
- Pablo Roldán Gómez, J. Berthold, Rita Loogen: *Eden Trace Viewer: Visualizing Parallel Functional Program Execution*, September 2005, submitted.
- Abyd Al Zain, Jost Berthold, Hans-Wolfgang Loidl: *A Generic Parallel Runtime-environment for High Performance Computation on Wide Area Networks*, in preparation.

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 meta-programming facilities in GHC.
- Jost and Rita continue working on the skeleton library.

Further reading

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

3.2.5 HCPN – Haskell-Coloured Petri Nets

Report by:	Claus Reinke
Status:	no news

Haskell-Coloured Petri Nets (HCPN) are an instance of high-level Petri Nets, in which anonymous tokens are replaced by Haskell data objects (and transitions can operate on that data, in addition to moving it around).

This gives us a hybrid graphical/textual modelling formalism for Haskell, especially suited for modelling concurrent and distributed systems. So far, we have a simple embedding of HCPN in Haskell, as well as a bare-bones graphical editor (HCPN NetEdit) and simulator (HCPN NetSim) for HCPN, building on the portable wxHaskell GUI library (→4.5.1). The tools allow to create and modify HCPN, save and load models, or generate Haskell code for graphical or textual simulation of HCPN models. HCPN NetEdit and NetSim are not quite ready for prime time yet, but functional; as long as you promise not to look at the ugly code, you can find occasionally updated snapshots at the project home page, together with examples, screenshots, introductory papers and slides.

This is still a personal hobby project, so further progress will depend on demand and funding. In other words, please let me know if you are interested in this!

Further reading

◦ Project home:
http://www.cs.kent.ac.uk/~cr3/HCPN/
◦ Petri Nets home:
http://www.informatik.uni-hamburg.de/TGI/PetriNets/

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.

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. There are ongoing efforts to develop the ideas in Edwin Brady's PhD thesis about efficiently compiling dependently typed programming languages.

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/.

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 monad transformers, 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 are currently in the process of cabalizing (→ 4.1.1) our code and moving to the darcs version control system (→ 6.6).

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

Report by:	Martin Sulzmann
Participants:	Gregory J. Duck, Simon Peyton Jones, Edmund Lam, Peter J. Stuckey, Martin Sulzmann, Peter Thiemann, Jeremy Wazny
Status:	on-going

Latest developments:

Extended algebraic data types

Extended algebraic data types subsume generalized algebraic data types (GADTs) and type classes with existential types. They are implemented in Chameleon and allow to express sophisticated program properties.

Co-induction and type improvement in type class proofs

We have already formalized and are currently implementing a significant extension of the dictionary-translation scheme for type classes where type class proofs may make use of co-induction. Our translation scheme extends to deal with type improvement as found in systems of functional dependencies and associated type synonyms.

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. A prototype implementation, examples and a number of papers can be 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) (→ 2.6) 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 Grammer 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/groups/ST/twiki/bin/view/Center/AttributeGrammarSystem
- Parser combinators:
  http://www.cs.uu.nl/groups/ST/Software/UU_Parsing/

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.

Preprocessors

DrIFT is a preprocessor which generates instances of generic functions. It is used in Strafunski (→ 4.3.3) to generate a framework for generic programming on terms. New releases appear regularly, the latest is 2.1.2 from September 2005.

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 ICFP '05) *Scrap your boilerplate with class: extensible generic functions*. This paper shows how you can turn 'closed' definitions of generic functions (not extensible when new data types are defined) into 'open', extensible, definitions.

The SYB approach to generic programming is used by James Cheney in his ICFP '05 paper *Scrap your nameplate*, in which he shows how to deal with names, name-binding, and fresh name generation generically.

Ralf Hinze has turned his ICFP 2004 paper *Generics for the masses* into a journal paper, which is to appear in the special issue of Journal of Functional Programming on ICFP 2004. The paper shows how you can do a lot of generic programming already in Haskell 98 (without extensions).

The paper *TypeCase: A Design Pattern for Type-Indexed Functions* (Bruno Oliveira and Jeremy Gibbons, Haskell Workshop 2005) explains some of the techniques behind the so-called "lightweight approaches to generic programming" of Cheney and Hinze, and shows their use in some other contexts.

Generic Haskell In *Type Inference for Generic Haskell*, Alexey Rodriguez et al. show how to infer the base type for a restricted class of generic functions.

Alimarine et al. show how to implement invertible arrows in Generic Haskell in *There and Back Again – Arrows for Invertible Programming* (HW 2005).

Other Artem Alimarine defended his PhD thesis *Generic Functional Programming* in Nijmegen, The Netherlands, in September 2005. In this thesis he shows how to extend Clean with a generic programming construct. The extension is similar to the Derivable Type Classes approach in Haskell, but has been worked out to a greater extent. Alimarine shows, amongst others, how to optimize the code generated for instances of generic functions. A similar approach to code optimization has been used in Generic Haskell as well. The generic extension in Clean has been used, amongst others, to develop generic editors, a generic testing framework, and generic web applications. See http://www.cs.ru.nl/st/Onderzoek/Publicaties/publicaties.html for some recent papers on these applications of generic programming.

In the *Lazy Polytypic Grid* project Colin Runciman and collaborators intend to investigate the combination of generic programming with staged computation to develop visualization algorithms and applications that can be adapted to specific data representations and computational resources, and how, coupled with the use of demand-driven evaluation, these technologies can provide new ways of managing the visualization of large datasets.

Jeremy Gibbons presented a tutorial *Design Patterns as Higher-Order Datatype-Generic Programs* at ECOOP (Glasgow, July 2005) and OOPSLA (San Diego, October 2005).

Justin Ward, Garrin Kimmell, Perry Alexander present a paper entitled *Prufrock: A Framework for Constructing Polytypic Theorem Provers* at the ASE 2005.

Current Hot Topics

Generic Haskell: finding transformations between data types. Adding type inference and views 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.

Major Goals

Efficient generic traversal based on type-information for premature termination (see the Strafunski project (→ 4.3.3)). Exploring the differences in expressive power between the lightweight approaches and the language extension(s).

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/

There is a mailing list for Generic Haskell: ⟨generic-haskell@generic-haskell.org⟩. See the homepage for how to join.

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.

Status

We have made a 1.0 release of the first phase, Cabal, the common build system. Cabal is now distributed with GHC 6.4, Hugs March 2005, and nhc98 1.18. Layered tools have been implemented, including `cabal2rpm` and `dh_haskell` (→ 7.4.1), for building Redhat and Debian packages out of Cabal packages. All of the fptools tree has been converted to using Cabal, as well as many other tools released over the last few months. Since the 1.0 release, many features have been added including support for profiling.

HackageDB, authored by Lemmih, is in a prototype phase. Users can upload tarred-and-gzipped packages to the database, and HackageDB will unpack them and make them available for clients via the XML-RPC (→ 4.7.7) interface. The prototype client, cabal-get, can download and install a package and its dependencies.

Further reading

- http://www.haskell.org/cabal
- http://hackage.haskell.org

4.1.2 Eternal Compatibility in Theory – a module versioning protocol

Report by: Sven Moritz Hallberg

I've spent some thought on module versioning, i.e. how to avoid module breakage when external dependencies change their interface in newer versions. I think I've come up with a nice and simple solution which has been published in an article for The Monad.Reader (→ 1.5). Here's the short intro:

As a program module evolves, functions and other elements are added to, removed from, and changed in its interface. It is clear that programs importing the module (it's dependants) will not be compatible with all versions. At least, each program is compatible with one version, the one the author originally used, and usually a few ones before and after that. But if a program is not continuously updated, with time, chances rise dramatically that one of it's dependencies as installed on a given host system will be incompatible. Alas, the program cannot be used. This effect comprises a major source of bit rot. To avoid such a situation, I suggest, in short, to append version numbers to module names, retaining the original name as a short-hand for "latest version".

For the complete description, please see the article linked to below. It describes the scheme which I have dubbed "ECT" in detail, as a protocol to be followed by the module implementor. For what it's worth, I have already adapted my own module System.Console.Cmdline.Pesco (→ 4.2.7) to use it.

If you are a module author, please have a look, tell me what you think, and consider adopting the ECT scheme yourself.

Further reading

http://www.haskell.org/tmrwiki/EternalCompatibilityInTheory

4.2 General libraries

4.2.1 LicensedPreludeExts

Report by:	Shae Erisson

The PreludeExts wiki page started with an oft-pasted email on the `#haskell` IRC channel (→ 1.2), where at least once a week someone asked for a permutations function. That sparked a discussion of what code is missing from the Prelude, once the wiki page was started, submissions poured in, resulting in a useful and interesting collection of functions. Last year's PreludeExts has become this year's BSD LicensedPreludeExts since John Goerzen wanted to have explicit licensing for inclusion into debian packages. If you contributed code to PreludeExts and haven't yet moved it to LicensedPreludeExts, please do so!

http://www.haskell.org/hawiki/LicensedPreludeExts

4.2.2 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://scannedinavian.org/~lemmih/hacanon-light

4.2.3 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 (→ 4.2.2) to simplify the binding process and Zeroth (→ 5.5.2) to avoid linking with Template Haskell.

Further reading

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

4.2.4 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.

Further reading

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

4.2.5 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/

4.2.6 Process

Report by:	Bulat Ziganshin
Status:	beta

Process is a fun library for easing decomposition algorithms to several processes, which transmit intermediate data via Unix-like pipes. You can write, for example:

```
  runP $ producer |> transformer1
                  |> transformer2
                  |> printer
```

where each "sub-process" in transporter is just a function started with `forkIO/forkOS` with one additional parameter-pipe. This pipe can be "read" with the `readP` function to get data from previous process in transporter, and "written" with `writeP` to send data to next process. A pipe can be made one-element (`MVar`) with the `|>` operator, or multi-element (`Chan`) with `|>>>`. Also supported are "back pipe" which can be used to return to previous process acknowledgements or, for example, borrowed buffers. Processes or entire transporters can also be run asynchronously and then communicated via a returned pipe:

```
  pipe <- runAsyncP $
          transformer1 |> transformer2
```

Moreover, processes/transporters can be run against four functions, which will be used for all it's piping operations. That opens a whole range of possibilities to create more complex process-control structures.

This lead to situation when Process, while more a syntactic sugar for well-known `forkOS/MVar/Chan` ingredients, than a "real" library, has become a very useful tool for assembling complex algorithms from simple pieces, which somehow transform data. This is like the situation of Unix popularity because it provides the same instruments for assembling together separate simple programs, but in this case you don't transmit plain byte streams, but typed data.

Further reading

- Download page: http://freearc.narod.ru

4.2.7 System.Console.Cmdline.Pesco – a command line parser ≠ GNU getopt

Report by:	Sven Moritz Hallberg
Status:	active development

My command line parsing module first reported in the November issue has just been updated to version 2. This is mainly a restructuring release. I've changed the module name from Pesco.Cmdline to System.Console.Cmdline.Pesco, to better fit into the overall hierarchical module namespace. Also the release now comes as a nice Cabal package (→ 4.1.1).

The code itself has been adapted it to use the ECT versioning scheme (→ 4.1.2) and has seen the addition of a minor but very convenient feature. In particular, the standard off-the-mill command line tool can now be written in a form like the following.

```
import System.Console.Cmdline.Pesco_2

-- command line option specifications
opts = [ flag ["bar"] "behave like Bar(1)"
         {-...-}
       ]
-- names for mandatory non-option arguments
args = [ "file1", "file2" ]

main = do Args parm nonopts
            <- stdargs "Foo" "1.0"
               "Do the foo-foo dance."
               opts args
          let [file1,file2] = nonopts
          if (parm "bar")
              then putStrLn "--bar given"
              else return ()
          {-...-}
```

The above program will then accept usage of the form

```
./Foo [options] file1 file2
```

where options can be `--bar` etc. Most importantly, it will automatically support the standard `--help` and `--version` flags and check if the required number of non-option arguments is present.

The module is available as a Cabal package named `pesco-cmdline`. It, and all associated documentation can be found on the website below, under the heading "System.Console.Cmdline.Pesco".

As of yet, the module still does not support explicitly reporting errors, it always calls `error`. Also, it is still not possible to ignore unrecognized command line arguments (for chaining command line parsers) or errors in general. These points will be addressed in the next major revision.

Further reading

http://www.scannedinavian.org/~pesco/

4.2.8 TimeLib

Report by:	Ashley Yakeley
Status:	active development

TimeLib is an attempt to redesign the current library for handling time (System.Time), balancing expressive functionality and intelligible simplicity. Now at version 0.2, TimeLib features representation of TAI, UTC and UT1, as well as Gregorian, ISO 8601 week, and "year and day" calendars, time-zones, and functions for `strftime`-style formatting.

The source is in a darcs (→ 6.6) repository, and the API is viewable in haddock (→ 5.5.9) format.

Further reading

http://semantic.org/TimeLib/

4.2.9 The Haskell Cryptographic Library

Report by:	Dominic Steinitz

The current release (2.0.3) is reasonably stable with just some bugfixes to the build since the last report.

All contributions are welcome.

Further reading

http://www.haskell.org/crypto

4.2.10 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 (→ 5.4.4) properties, superfluous superclasses 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. Both new types (like power series and values with physical units) and type classes (like the `VectorSpace` multi type class) are introduced. Using the revised system requires hiding some of the standard functions provided by Prelude, which is fortunately supported by GHC.

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, which will probably lead to mutual recursive dependencies.

A problem which is still not solved satisfyingly arises e.g. for residue classes and linear algebra. 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://cvs.haskell.org/darcs/numericprelude/

4.2.11 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.

The monad library is still alive and I am using it for my projects. I am not aware of any other users. For changes and the most recent version one can visit its website (see below).

Soon there will be a new release (1.5), the main changes of which are:

- A new API to the backtracking transformer (now called SearchT), inspired in part by one of the this year's ICFP papers.
- Reverted to having only one base monad Id (plus the usual Haskell IO,ST,etc), all other monads can be constructed by applying the appropriate transformer. This makes the library smaller, and hopefully easier to understand and maintain.
- Some of the transformers are more strict in their internal structures (StateT and WriterT in particular).
- More examples of how to use the transformers, which also serve as test cases.

Further reading

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

4.2.12 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 0.9.10 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/

4.2.13 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

4.2.14 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

4.2.15 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 filesystem 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

4.2.16 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 (→ 4.2.15)). 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 te 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 Haskell-Source with eXtensions (HSX, haskell-src-exts)

Report by:	Niklas Broberg
Status:	beta

HSX aims to be a replacement of the libraries in Language.Haskell of the standard haskell-src package. The contribution is that HSX supports a good deal of the various syntactic extensions available, such as
- Multi-parameter type classes with functional dependencies
- Empty data declarations
- GADTs
- Implicit parameters (ghc and hugs style)
- Template Haskell (broken for 6.4, needs redoing)

Apart from these standard extensions, it also handles regular patterns as per the HaRP (→ 3.1.8) extension as well as HSP-style embedded XML syntax (→ 3.1.6).

Further reading

- Webpage and darcs repo at:
  http://www.cs.chalmers.se/~d00nibro/haskell-src-exts/

4.3.3 Strafunski

Report by:	Joost Visser
Status:	active, maintained, new release in November 2005
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 (→ 5.2.7) 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.

The Sdf2Haskell component of Strafunski has recently been extended to offer not only parsing support via the external "sglr" parser, but also:

- parsing support via HaGLR, an experimental 100% Haskell implementation of Generalized LR parsing
- pretty-printing support, based on the pretty-print combinators as available in the Text.PrettyPrint.HughesPJ library. The generated pretty-printers are functional strategies that offer uniform behaviour which can be customized with type-specific behaviour.

Strafunski is used in the HaRe project (→5.3.3) and in the UMinho Haskell Libraries and Tools (→7.3.9) to provide analysis and transformation functionality for languages such as Java, VDM, SQL, spreadsheets, and Haskell itself.

Further reading

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

4.4 Data handling

4.4.1 Hierachical Libraries Collections (formerly DData)

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

The standard collections data structures have recently been replaced by (a modified version of) Daan Leijen's DData library.
Yet, many people would like them futher improved. Maintenance continues, with the goal to reach greater quality standards.

Further reading

http://haskell.org/hawiki/StandardCollectionLibraries/

4.4.2 fps (fast packed strings)

Report by:	Don Stewart
Status:	active development

FPS provides packed strings (byte arrays held by a ForeignPtr), along with a list interface to these strings. This library provides a faster and wider range of operations than that found in the standard PackedString library, and is a port of the packed string code found in darcs. Such strings typically reduce the time and space requirements of a similar program based on `[Char]`. It also lets you do extremely fast IO in Haskell using mmap; in some cases, even faster than typical C implementations.

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

4.4.3 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 (→5.4.4)), 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

4.4.4 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.

The HList library has been extended to enable all recent OOHaskell examples and improve their efficiency. We added more specialized but more efficient implementations of the following Record operations: extension, field lookup, projection. These operations now have simpler types (with fewer constraints), which notably simplifies the inferred types of OOHaskell programs.

We added a function to narrow two records to their automatically computed least-upper bound. We defined functions `nilLub` and `consLub` to construct a *homogeneous* list out of heterogenous record components. The component records are automatically narrowed to their least-upper bound.

We are working on Cabalizing (→ 4.1.1) HList, expanding on the work by Einar Karttunen.

Further reading

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

4.4.5 Takusen

Report by:	Alistair Bayley, Oleg Kiselyov, Alain Crémieux
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 and Sqlite (including bind variables and pre-fetching), and partially support PostgreSQL.

Since the last report we have added a back end for PostgreSQL. Bind variables are not supported yet in this case, however. We now distinguish nulls from empty strings, provided the database does so. We have restructured the tests to more cleanly separate database-dependent and independent layers. We are working on a cleaner interface for prepared statements, and a MS Sql Server implementation.

Further reading

http://cvs.sf.net/viewcvs.py/haskell-libs/libs/takusen/

4.4.6 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 (→ 4.5.1). 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 (→ 4.2.12)

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/

4.4.7 ByteStream

Report by:	Bulat Ziganshin
Status:	beta

ByteStream is like the NHC Binary library – it provides marshalling of Haskell objects to byte streams and restoring them back. Features:

- light-fast speed, but only x86 processors compatible (uses unaligned memory access)
- using callbacks to read and write data (in large chunks) to a byte stream, so it can go on-the-fly to memory, file or, for example, to another PC
- using variable-length format for Integers and list lengths (1–9 bytes, dependent on value)

  Example of very basic usage:

```
ByteStream.writeFile "test" [1..1000::Integer]
(restored::[Integer]) <- ByteStream.readFile "test"
```

Further reading

- Download page: http://freearc.narod.ru

4.4.8 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.

Further reading

- Download page: http://freearc.narod.ru

4.5 User interfaces

4.5.1 wxHaskell

Report by:	Daan Leijen
Status:	beta

wxHaskell is a portable GUI library for Haskell. The goal of the project is to provide an industrial strength portable GUI library, but without the burden of developing (and maintaining) one ourselves.

wxHaskell is therefore build on top of wxWidgets – a comprehensive C++ library that is portable across all major GUI platforms; including GTK, Windows, X11, and MacOS X. Furthermore, it is a mature library (in development since 1992) that supports a wide range of widgets with native look-and-feel, and it has a very active community (ranked among the top 25 most active projects on sourceforge). Many other languages have chosen wxWidgets to write complex graphical user interfaces, including wxEiffel, wxPython, wxRuby, and wxPerl.

Since most of the interface is automatically generated from the wxEiffel binding, the latest release of wxHaskell already supports about 90% of the wxWindows functionality – about 3000 methods in 500 classes with 1300 constant definitions. wxHaskell has been built with GHC 6.x on Windows, MacOS X and Unix systems with GTK, and binary distributions are available for common platforms.

Since the last community report, most work has been directed into improved stability and a better build system. There is also better integration with other packages: HaskellDB (→ 4.4.6) works with the wxHaskell ODBC binding and HOpenGL (→ 4.6.1) can work with the OpenGL canvas. The wxHaskell website also shows some screenshots of larger sized applications that are developed with wxHaskell. It is most satisfying to see that even those larger applications are ported without any real difficulties – Haskell is becoming a very portable language indeed!

Further reading

You can read more about wxHaskell at http://wxhaskell.sourceforge.net and on the wxHaskell mailing list at http://sourceforge.net/mail/?group_id=73133. See also "wxHaskell: a portable and concise GUI library", Daan Leijen, Haskell workshop 2004.

4.5.2 FunctionalForms

Report by:	Sander Evers

FunctionalForms is a combinator library/domain specific language for *forms* in wxHaskell (→ 4.5.1): dialogs which show and edit a set of values (often named *Options* or *Settings*). This declarative abstraction layer combines control and layout definition into one expression, and passes values to and from the controls. Disjoint union types directly translate to radiobuttons or checkboxes with subforms. All of wxHaskell's layout freedom is preserved.

Currently, FunctionalForms is mainly a proof-of-concept for declarative form programming. Plans exist to integrate it more closely with wxHaskell.

Further reading

http://www.sandr.dds.nl/FunctionalForms

4.5.3 Gtk2Hs

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

Gtk2Hs is a GUI Library for Haskell based on Gtk+. Gtk+ is an extensive and mature multi-platform toolkit for creating graphical user interfaces.

GUIs written using Gtk2Hs follow the native look on Windows and of course 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
- extensive reference documentation
- 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 rendering engine
- 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 have just released version 0.9.10. It includes many bug fixes, various documentation improvements and two significant new features:

- Addition of the cairo vector graphics API. Paolo Martini won a $4500 grant under the Google Summer of Code programme for a project to add bindings for the cairo vector graphics library to Gtk2Hs. The result of this project are now included in the latest Gtk2Hs release. This provides a easy-to-use vector graphics API (using a PDF-style drawing model) with high quality output for multiple backends (screen, print and image files).

- Completion of Pango, the font rendering engine of Gtk+. We now provide all functions that end users should ever need for rendering text, ranging from type-setting whole paragraphs (Layouts) down to breaking up attributed text into several runs. All offsets into Haskell strings are transparently translated into UTF-8 offsets used in Pango, thereby alleviating much of the grief of dealing with Unicode.

This release of Gtk2Hs is known to run on Windows, Linux, MacOS X, FreeBSD, OpenBSD and Solaris.

To go with the 0.9.10 release there is an implementation of the SOE graphics API. It takes advantage of the cairo vector graphics extension if it is available to produce anti-aliased output. The intention is to bundle this with a future release of Gtk2Hs so that it will be easy for students to install on Windows and Linux.

Other news since the last HCAR:

- Gtk2Hs now has a properties API in the same style as that of wxHaskell (→ 4.5.1)/Yampa.

- The new Pivotal (→ 3.1.3) prototype uses Gtk2Hs. Pivotal is an interactive, document-centered presentation of Haskell.

- The Yi (→ 6.13) text editor and Haskell IDE (→ 5.5.6) are now using Gtk2Hs

- Gtk2Hs is being used on Functional Programming courses at the University of Oxford and the University of Jyväskylä.

- The development version of Gtk2Hs is now available via darcs (→ 6.6). We welcome patches contributed using "darcs send".

Future plans include a 1.0 release, bundling the SOE package, HOpenGL (→ 4.6.1) support and writing some introductory tutorials.

Further reading

- News, downloads and documentation: http://haskell.org/gtk2hs/
- Development version: `darcs get` http://haskell.org/gtk2hs/darcs/gtk2hs/
- Graphics.SOE.Gtk implementation: `darcs get` http://haskell.org/~duncan/soe

4.5.4 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/

4.6 (Multi-)Media

4.6.1 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 (→ 4.5.1) 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 planned
- 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.

The binding comes with a lot of 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/

4.6.2 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 spiri", 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 have not been released yet, but they are already usable and almost feature-complete. They cover the latest specification releases, i.e. OpenAL 1.1 and ALUT 1.0.5, 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/

4.6.3 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://scannedinavian.org/~lemmih/hsSDL/
- libSDL:
  http://www.libsdl.org/

4.6.4 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. The `Music` data type has been generalized in the style of Hudak's "polymorphic temporal media."

2. The `Music` data type has been made abstract by providing functions that operate on it.

3. Support for infinite `Music` objects is improved.

4. 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 (→ 6.20).)

5. The test suite is now based on QuickCheck (→ 5.4.4) and HUnit.

6. The AutoTrack project has been adapted and included.

7. Support for csound orchestra files has been improved and extended, thus allowing instrument design in a signal-processing manner using Haskell.

Future plans

Introduce a more general notion of instruments which allows for more parameters that are specific to certain instruments. Allow modulation of music similar to the controllers in the MIDI system. Connect to other Haskore related projects. Adapt to the Cabal (→ 4.1.1) system.

Further reading

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

4.7 Web and XML programming

4.7.1 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 (→ 4.7.4) to query search engines and parse HTML responses.

Current Status

The current version of CabalFind is 0.1
Number of Cabal package descriptor files that may be found by CabalFind as of October 25, 2005 using Google is 117 (results may vary slightly from time to time). Ironically, CabalFind cannot find its own Cabal package descriptor file (`CabalFind.cabal`): reason yet unknown.

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.

4.7.2 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 (→4.7.5) is used. HaskellDB (→4.4.6) 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

4.7.3 HaXml

Report by:	Malcolm Wallace
Status:	stable, maintained

HaXml provides many facilities for using XML from Haskell. The public release was recently refreshed to 1.13, mainly for compatibility with ghc-6.4, and to introduce support for building via Cabal (→4.1.1).

I have recently been experimenting with improvements to the secondary parsing stage, where the generic XML tree is re-parsed into typed Haskell trees. Until now, there have been two ways of doing this, depending on whether you started by defining a DTD (Xml2Haskell) or started with some Haskell types (Haskell2Xml). These can be combined into a single class, and what is more, they can be implemented with proper parser combinators, to provide decent error messages. This experimental and incomplete branch of HaXml is version 1.14, and once it is stable, it will become version 2. Work still remaining is to update the tools DtdToHaskell and DrIFT (→3.4) such they the generate the new class rather than the old ones.

Further reading

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

4.7.4 Haskell XML Toolbox

Report by:	Uwe Schmidt
Status:	fourth major release (current release: 5.3)

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 (→4.7.3) 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 developping a dynamic webserver with servlet functionality. XML and HXT will be used for all internal data. The server will be based on HWS-WP (Haskell Webserver with Plug-Ins).

Further reading

The Haskell XML Toolbox Webpage (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.

4.7.5 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:
- restructuring towards Cabalization (hierarchical modules, one package)
- 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 (→ 4.2.12)) as well as the implementation of a session as a continually running server thread

Current work includes
- database interface
- authentication interface
- user manual (still in the early stages)

Further reading

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

4.7.6 HAIFA

Report by: Simon Foster

My work on GXS since the last HCAR has primarily been to move away from the type-safe cast method of building generic functions, toward the new extensible type-class based SYB3 library (→ 3.4). GXS is now fully extensible, and allows full customization of data-type encoders, as well as the addition of hooks, which allows additional meta-data to be encoded into the tree.

To facilitate the use of W3C XML Schema for mapping Haskell data-types we've also been extending the content-model of GXS, to be suitably expressive. We've utilized Ralf Lämmel's HList (→ 4.4.4) library to build representations of type-based Union and Sequences, to allow a natural representation of data-types encoded by Schema. With the use of a newly implemented set of data-types for representing XML Schema, it is now possible to map Schema complex-types to Haskell data-types, with full serialization, although this highly beta at the moment.

All of this has been moving toward the use of Haskell for orchestrating composite web-services. One of our aims is to allow Haskell code to be evaluated via a Web-Service, with inputs and outputs to a function abstraction encoded as XML, and typed by XML Schema. We have successfully been able to build the service to perform this task, and will shortly be releasing the code under the GPL. As well as this, we have started putting together the actual orchestration engine, which uses a process calculus to provide operational semantics for the workflow. This too will hopefully be released soon.

No further work has been done on HWS-WP, mainly because we are now using a much simpler HTTP server as our shell, which is part of HAIFA. Our SOAP implementation is also usable server-side.

Further reading

For more information please see the HAIFA project page at http://savannah.nongnu.org/projects/haifa or the HAIFA Wiki at http://www.repton-world.org.uk/mediawiki/index.php/HAIFA_Wiki.

4.7.7 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, minor improvements over time

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 (→ 5.1.2), hacanon.

Current Status

The current release version of hsffig is 1.0pl2 (release date: October 17, 2005).

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/
- The latest addition to the Tutorial:
  http://haskell.org/hawiki/HsffigLinkageOptimization/
  This page describes a method to optimize the process of linking with the object code resulting from compilation of Haskell code produced by HSFFIG, and shows how to use the `--make` option of GHC when building applications using such code.

5.1.2 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 latest version of C–>Haskell includes support for cross compilation, a completely new Cabal-ised (→ 4.1.1) build-system, and library-less bindings that are easier to distribute. Moreover, Duncan Coutts dramatically lowered time and memory consumption with a new C parser. 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

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:

- We are switching to a Cabal build system (→ 4.1.1) for Alex, which will make it easier to build Alex on Windows systems. Alex needs some features found only in very recent versions of Cabal, however.
- Alex is now in a Darcs repository (→ 6.6), here: http://cvs.haskell.org/darcs/alex. The version of the code in Darcs is currently not buildable, because we are in the progress of migrating to Cabal.

Further reading

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

5.2.3 Happy

Report by:	Paul Callaghan and Simon Marlow
Status:	stable, maintained

Paul's Generalized LR (GLR) extension for Happy was released as part of Happy-1.15. This release also includes some new directives and some fixes, plus Ashley Yakeley has modified the monad mode of standard (LALR) parsers to carry additional class constraints. To fit in with this last change, parsers which don't have a monad specified will now be generated to use an identity monad.

Based on an algorithm by Tomita, GLR can parse ambiguous grammars and produce a directed acyclic graph representing all possible parses. It is based on undergraduate project work by Ben Medlock, but has been significantly extended and improved since then. You can also attach semantic information to rules in two modes:
- to give detailed, application-specific labelling for the nodes in the DAG;
- to compute lists of overall semantic results, one per valid parse.

The latter mode can also perform monadic computations. We have used the GLR facility in several applications, including analysis of DNA sequences and determination of correct rhythmic structures for poetry. Other possible applications include natural language and pattern analysis. The Chalmers BNFC tool (→ 5.2.5) is able to output grammars which are compatible with the GLR mode of Happy. Recently, the driver code has been improved and is significantly faster on heavily ambiguous grammars, although this is not part of the official release at present (see Paul's GLR page or CVS to obtain a copy).

Other current activity on Happy:

- We are switching to a Cabal build system (→ 4.1.1) for Happy, which will make it easier to build Happy on Windows systems. Happy needs some features found only in very recent versions of Cabal, however.
- Happy is now in a Darcs repository (→ 6.6), here: http://cvs.haskell.org/darcs/happy. The version of the code in Darcs is currently not buildable, because we are in the progress of migrating to Cabal.

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, so I will be working on chasing out the bugs and submitting a patch for inclusion in the official Happy distribution.

Further reading

- There is a darcs repo (→ 6.6) based on the Happy 1.15 source distribution at: http://www.eecs.tufts.edu/~rdocki01/happy-ag/
- Documentation for the extension can be found at: http://www.eecs.tufts.edu/~rdocki01/happy-ag-docs/sec-AttributeGrammar.html

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 LRC

Report by:	Joost Visser

Lrc is a system for generating efficient incremental attribute evaluators. Lrc can be used to generate language based editors and other advanced interactive environments. Lrc can generate purely functional evaluators, for instance in Haskell. The functional evaluators can be deforested, sliced, strict, lazy. Additionally, for easy reading, a colored LaTeX rendering of the generated functional attribute evaluator can be generated. Recently, a front-end has been added to Lrc for XQuery.

5.2.7 Sdf2Haskell

Report by:	Joost Visser

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.

Recently, the functionality of Sdf2Haskell has been extended with generation of pretty-print support. From the SDF grammar, a set of Haskell functions is generated that defines an 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 (→ 4.3.3). Sdf2Haskell has recently been used in the development of a parser and pretty-printer for the complete ISO standard VDM specification language (in the context of VooDooM (→ 5.3.4)).

5.2.8 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 (→ 7.3.9).

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, 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

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

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

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 (→ 4.5.1).

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

- Crossing the Rubicon: from Haskell to .NET through COM. ERCIM News 63:51-52, October 2005. http://www.ercim.org/publication/Ercim_News/enw63/lucas.html

- 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, 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 6.4.1.

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 (→ 5.5.9) documentation. Please let us know if you are using the API.

Our immediate aims are to support more data-oriented refactorings and to support duplicate code elimination and function slicing. We are actively exploring how to make it easier to use HaRe with GHC and its libraries, and we have recently undertaken a feasibility study on this.

A snapshot of HaRe is available from our web page, as are recent presentations from the group (including LDTA 05, TFP), 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).

The final report for the project appears there too, together with an updated refactoring catalogue and the latest release of the system. Coming in early 2006 will be Huiqing's PhD thesis, which was submitted in September.

Further reading

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

5.3.4 VooDooM

Report by:	Joost Visser
Maintainer:	Tiago Alves, Paulo Silva

VooDooM reads VDM-SL specifications and applies transformation rules to the datatypes that are defined in them to obtain a relational representation for these datatypes. The relational representation can be exported as VDM-SL datatypes (inserted back into the original specification) and/or SQL table definitions (can be fed to a relational DBMS). The first VooDooM prototype was developed in a student project by Tiago Alves and Paulo Silva. Currently, the development of VooDooM is continued as an open source project (http://voodoom.sourceforge.net/) in the context of the IKF-P project (Information Knowledge Fusion, http://ikf.sidereus.pt/) and will include the generation of XML and Haskell.

Further reading

VooDooM is available from http://voodoom.sourceforge.net/. The implementation of VooDooM makes ample use of strategic programming, using Strafunski (→ 4.3.3), and is described in *Strategic Term Rewriting and Its Application to a VDM-SL to SQL Conversion* (Alves et al., Formal Methods 2005).

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.

In May 2005 Hat developers from the University of York and the University of Kent met in York for a Hat-Day. The current work in progress, from sketchy ideas to recently implemented extensions, was discussed. In particular, the extended algorithmic debugger by Thomas Davie, hat-delta, is now available in CVS and Tom Shackell also showed at TFP 2005 how the Hat G-machine enables faster production of redex trails.

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 prototype viewing tools 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.
- Tom Shackell and Colin Runciman: *Faster production of redex trails: The Hat G-Machine.* Trends in Functional Programming, TFP '05, Symposium proceedings.
- http://www.haskell.org/hat

5.4.3 buddha

Report by:	Bernie Pope
Status:	active

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.

New features are being implemented and tested. It is not clear when a stable release will emerge. Perhaps early 2006.

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).

Further reading

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

5.4.4 QuickCheck

Report by:	Koen Claessen and John Hughes
Status:	active development

QuickCheck is a tool for specifying and testing formal properties of Haskell programs. There have been several inofficial draft versions of QuickCheck around.

Right now we are in the process of packaging up a new, official version of QuickCheck, integrating support for:

- automatic finding of small counter examples
- monadic properties
- exception handling and time-outs
- stating properties that are expected to fail
- a callback hook for displaying failing test cases
- generating test reports

And lots lots more! We plan to distribute the new QuickCheck using the new Haskell Cabal (→ 4.1.1).

An accompanying tutorial, explaining typical problems and programming idioms that solve them is also in the make.

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, o r nhc98 (→ 2.3) – except jhc (→ 2.5) (which does not compile modules separately anyway). The public release is currently version 3.10. 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://scannedinavian.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 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 1.1, and is pretty stable – there have been only very minor bugfixes in the last couple of releases.

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, verison 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.

Help is (still) welcome! You first need to register for the Microsoft VSIP (Visual Studio Integration Program) to get access to the VSIP SDK, which has tools, APIs and documentation for extending Visual Studio. Registering for VSIP is free, but you have to agree to a longish license agreement: http://www.vsipdev.com/.

If you've registered for VSIP and would like to contribute to Visual Studio/Haskell, please drop me a note (Simon Marlow ⟨simonmar@microsoft.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) (→ 1.2) 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 (considered 'alpha'). It features a project model, a configurable source code editor (with syntax coloring and code assist), compiler support for GHC, interpreter support for GHCi and HUGS, documentation generation with Haddock (→ 5.5.9), and launching from the IDE. In the time between the last HC&A report and now we have experimented in a number of different directions, including Debugging support using the Eclipse Debug framework and the possibility to write part of the Haskell plugins in Haskell itself. So far we are doing this with lots of handcoding using JNI and FFI. We are trying to find a way to simplify and generalize this so that Eclipse plugins can partly be written in Haskell. We have also started to add some design documents about this to the project homepage.

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 haste

Report by:	Rickard Nilsson

Haste – Haskell TurboEdit – is an integrated development environment for Haskell, written in Haskell. It is built on the wxHaskell GUI library (→ 4.5.1), and currently runs on Linux and Windows. It features project management, syntax highlighting of Haskell code, code completion functionality, and integration with GHC and GHCi.

Haste was started as a school project by a group of undergraduate students at the CS department of Chalmers, Gothenburg. The intention is that development will continue – and hopefully attract more contributors – after it has finished as a school project, which will happen by end of May 2005.

An early alpha release of Haste was announced on April 10, 2005. In addition to building instructions for Linux, there exist a Windows installer and a Gentoo Linux package for Haste.

Further reading

http://haste.dyndns.org:8080

5.5.9 Haddock

Report by:	Simon Marlow
Status:	stable, maintained

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.

Current activity:

- We are switching to a Cabal build system (→ 4.1.1) for Haddock, which will make it easier to build Haddock on Windows systems.
- Haddock is now in a Darcs repository (→ 6.6), here: http://cvs.haskell.org/darcs/haddock. The version of the code in Darcs requires a very recent version of Cabal to build.

Further reading

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

5.5.10 Hoogle – Haskell API Search

Report by:	Neil Mitchell
Status:	v2.0, in progress

Hoogle is an online Haskell API search engine. It searches the functions in the standard 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, using unification. It also supports 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 and another full rewrite is under way. All the bugs have been ironed out and the user interface has been improved substantially. The current development goals are support for classes and instances, the `type` keyword, and full support for all of GHC's libraries. This upcoming release, Hoogle v3.0, should be made during November.

Hoogle now comes in many flavours, the original web interface, a command line tool, a lambdabot (→ 6.11) plugin, a firefox plugin and a GUI for the Mac.

Further reading

http://www.cs.york.ac.uk/~ndm/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`.

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:	Autrijus 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 (→ 4.2.12) package.

As of this writing, we are ramping up toward the 6.28.x milestone, with a unified object space for all four runtime backends. The next milestones will see Grammar support and a “dynamic when needed, static when possible” Type system.

The Pugs team has over 120 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

- Pugs homepage
  http://pugscode.org/
- Daily-updated development journal
  http://use.perl.org/~autrijus/journal/

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.

We are currently working towards a stable release of darcs version 1.0.4 [note by editor: released in the meantime], which features considerable improvements in performance and memory useage over version 1.0.3, plus a few new commands and interface improvements. We have had patches contributed by 28 different people since the 1.0.3 release, and more have helped by reporting bugs and good ideas on the mailing lists and bug tracking system.

Tomasz Zielonka stepped down as darcs-stable maintainer after the release of darcs 1.0.3, and this role was taken up by David Roundy for the release of darcs 1.0.4. After the release of darcs 1.0.4, the stable branch of darcs will be maintained by Tommy Pettersson. The unstable branch of darcs continues to be maintained by Ian Lynagh.

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 FreeArc

Report by:	Bulat Ziganshin
Status:	beta

FreeArc is an archiver program (like Info-ZIP). This class of programs is traditionally written in C/C++ (so-called “system programming”), so I was interested – how can Haskell compete with C++ in this field? By dividing the program in two parts – a computation-intensive compression library, written in C++, and all other code – working with lists of files, working with archive structure, interfacing with user – written in Haskell, I have got the resulting program competitive with archivers written in C++ (RAR, 7-zip, UHARC), while cutting development time by several times, and especially the number of errors made during development. Also, during development I have written several general-purpose Haskell libraries, which you can find in this Report (Compression Library (→ 4.4.8), ByteStream (→ 4.4.7), Process (→ 4.2.6)). You can download the program sources if you are interesting in replacing C++ with Haskell or developing general utilities with Haskell, and want to learn programming techniques suitable for this case.

The program sources are extensively commented . . . in Russian.

Further reading

- Download page: http://freearc.narod.ru

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.) (→ 4.7.7)
  http://www.bringert.net/haskell-xml-rpc/http.html

6.10 Hircules, an irc client

Report by: Jens Petersen

Hircules is a gtk2-based IRC client built on gtk2hs (→ 4.5.3) and code from lambdabot (→ 6.11). Currently it is not actively maintained: the last release is version 0.3, though there are some unreleased bug fixes and improvements that I should put out one day including a patch from Axel Simon to make it build with current gtk2hs. Contributions are very welcome.

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, pointfree programming, dictd client, fortune cookies, Google search, online help and more. Version 3 of lambdabot has been released, and development continues.

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 riot

Report by: Tuomo Valkonen

Riot is a tool for keeping (textual) information organised. Some people call such programs 'outliners'. It is a todo list and note manager, and a manager for whatever information one might collect. Riot has a curses-based interface resembling those of slrn and mutt and all text editing is done with your favourite external editor: Riot is just a nice-to-use browser and entry organiser for collections of text.

Further reading

The Riot homepage is at http://iki.fi/tuomov/riot/.

6.13 yi

Report by:	Don Stewart
Status:	active development

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. Work is now underway to provide a GTK gui, and to provide embedding support for Yi.

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 (→ 4.5.1) 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 (→ 4.5.1). 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 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/rosemeier.yarrow.en.html.

The new *Yarrow homepage* located at

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

Soon it will contain a copy of the homepage for the Haskell 1.3 version as well as the Haskell 98 adaption.

6.17 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.

Further reading

http://haskell.org/docon/

6.18 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 being prepared now.

Further reading

http://haskell.org/dumatel/

6.19 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.

There has not been a release for quite some time, but development continues slowly in the Subversion repository. Goals are to set up a library of useful formatting directives for inclusion in a future release, and to switch to Cabal (→ 4.1.1). Also, the `polytable` LATEX style that is used internally by lhs2TEX is getting a nicer user interface, possibly making it more widely applicable.

Further reading

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

6.20 Audio signal processing

Report by:	Henning Thielemann
Status:	experimental, active development

In this project audio signals are processed using pure Haskell code. This includes a simple signal synthesis backend for Haskore, filter networks, signal processing supported by physical units.

Future plans

Connect with the HaskellDSP library. Hope on faster code generated by some Haskell compilers. `:-)` Probably connect to some software synthesizer which is more efficient, but nearly as flexible as code entirely written in Haskell. Explore whether Monads and Arrows can be used for a more convenient structuring and notation of signal algorithms.

Further reading

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

6.21 Converting knowledge-bases with Haskell

Report by:	Sven Moritz Hallberg

In November 2004, I reported my writing a tool for research work which converts knowledge bases from a commercial tool (EngCon) to the LISP-based description language of our in-house tool (Konwerk).

The project, which was funded by the EU, is nearing its end and the converter tool has been updated with all major features we wanted, consisting of nearly 4000 lines of Haskell code. It will most likely graciously disappear into the eternal mist of time now. `:)`

In retrospect, Haskell provided a formidable vehicle for throwing up this program.

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/

7.2 Haskell in Education

7.2.1 Haskell in Education at Universidade de Minho

Report by:	Jorge Sousa Pinto

Haskell is heavily used in the undergraduate curricula at Minho. Both Computer Science and Systems Engineering students are taught two Programming courses with Haskell. Both programmes of studies fit the "functional-first" approach; the first course is thus a classic introduction to programming with Haskell, covering material up to inductive datatypes and basic monadic input/output. It is taught to 200 freshmen every year. The second course, taught in the second year (when students have already been exposed to other programming paradigms), focuses on pointfree combinators, inductive recursion patterns, functors and monads; rudiments of program calculation are also covered. A Haskell-based course on grammars and parsing is taught in the third year, where the HaLeX library is used to support the classes.

Additionally, in the Computer Science curriculum Haskell is used in a number of other courses covering Logic, Language Theory, and Semantics, both for illustrating concepts, and for programming assignments. Minho's 4th year course on Formal Methods (a 20 year-old course in the VDM tradition) is currently being restructured to integrate a system modeling tool based on Haskell and VooDooM. Finally, in the last two academic years we ran an optional, project-oriented course on Advanced Functional Programming. Material covered here focusses mostly on existing libraries and tools for Haskell, such as YAMPA – functional reactive programming with arrows, the WASH library, the MAG system, the Strafunski library, etc. This course benefitted from visits by a number of well-known researchers in the field, including Ralf Lämmel, Peter Thiemann, and Simon Thompson.

7.2.2 Functional programming at school

Report by:	Walter Gussmann
Status:	stable, maintained

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.

Beginners' course

In courses for beginners (2002/2003 – 18 pupils) you can use the functional qualities of Haskell: functions for logical gates, number conversions (bin2hex ...), function concatenation, simple list functions etc. can be build without writing much programming code.

Medium level courses

Last time when I taught pupils who had a one-year-experience of Pascal programming (2003/2004 – 12 pupils). 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 languages like Pascal or Java.

Advanced courses

Especially in high level courses the use of Haskell paid off. With 5 hours a week for 2 years these courses lead to the German "Abitur", ending with a 4-hour examination (2003–2005 – 11 pupils). I started the course with an introduction to Haskell and used Haskell until the end. We talked about recursion and recursive data structures with detailed examples like the Huffman-Tree (implemented for compressing text files). We also built op-trees to evaluate arithmetic terms and multi-trees to simulate virtual file systems. A highlight was the implementation of a module "turtle" based on Haskell's graphics library, with which the pupils created fractal pictures.

The last half year of the course (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.

What's new?

A few weeks ago 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.

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 Functional Programming at the University of Nottingham

Report by:	Joel Wright

Within the Foundations of Programming group at the University of Nottingham are a number of people working on Haskell related projects. These projects involve reasoning about Haskell programs, new Haskell features, and the implementation of new systems using Haskell. Members of the group are also involved in teaching Haskell, and computer aided formal reasoning using the Epigram (→ 3.3.1) system, to undergraduates. Specific research interests of the group include the following:

- Reasoning about synchronous and asynchronous exceptions in a simple language. This work was inspired by, and aims to reason about asynchronous exception combinators in, GHC. Some proofs using the Epigram system.
- Reasoning about the time requirements of functional programs, by counting evaluation steps that represent transitions in an underlying abstract machine. Work is also being carried out to extend this to measure space usage.
- Functional Reactive Programming in the Yampa system.
- Datatype Generic Programming (→ 3.4).
- The implementation of the dependently typed programming language, Epigram.
- A functional quantum programming language, QML: the compiler and simulator are implemented in Haskell.

Graham Hutton has also recently written a book (→ 1.6.1) aimed at teaching Haskell to students studying computing science at university level, but is also appropriate for a broader spectrum of readers who would like to learn about programming in Haskell.

Further reading

- Foundations of Programming Group page:
  http://www.cs.nott.ac.uk/Research/fop/index.html
- Functional Programming Blog:
  http://sneezy.cs.nott.ac.uk/fplunch/
- Programming in Haskell Book:
  http://www.cs.nott.ac.uk/~gmh/book.html
- Epigram:
  http://www.e-pig.org

7.3.2 Artificial Intelligence and Software Technology at JWG-University Frankfurt

Report by:	David Sabel
Members:	Matthias Mann, 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.

Ongoing work is devoted to develop applications using direct IO calls, i.e., using `unsafePerformIO` in arbitrary contexts. Another topic is the proof of correctness of further program transformations.

Non-deterministic Call-by-need Lambda Calculi

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. Current research is aimed towards extensions of this calculus, e.g. constructors and case or a recursive let.

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 decidable.

Implementations Using Haskell

As a final year project, *Christopher Stamm* implemented an 'Interpreter for Reduction Systems' (*IfRS*) in Haskell. IfRS is an interpreter for higher order rewrite systems that are based on structural operational semantics. Additionally, it is possible to define reduction contexts and to use contexts and domains (term sets that are definined similiar to contexts without holes) in the rewrite rules. Also, IfRS is able to test whether the reduction rules satisfy the conditions of the GDSOS-rule format. The GDSOS-rule format ensures that bisimulation is a congruence.

Current research topics of our group also encompass second order unification, higher order unification and context unification. It is an open problem whether (general) context unification is decidable. *Jörn Gersdorf* has implemented a non-deterministic decision algorithm for context matching in Haskell which benefits from lazy evaluation at several places.

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
- IfRS – Interpreter for Reduction Systems
  http://www.informatik.uni-frankfurt.de/~stamm

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 (→5.5.9), the combinator library Parsec, HaXml (→4.7.3) 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

One prong of the Metis Project at Brooklyn College, City University of New York, is research on and with Parallel Haskell in a Mosix-cluster environment.

We have implemented a new, contention-driven scheduler for GpH (→3.2.1) that we have demonstrated provides (at worst!) qualitatively improved performance under some circumstances, and we continue our work on debugging GUM in our version of Parallel Haskell.

Current programming efforts are focused on a multidimensional tic-tac-toe with an intelligent computer antagonist, and a Sudoku puzzle solver, both of which should provide good test cases for exercising the runtime system.

Further reading

http://www.sci.brooklyn.cuny.edu/~metis

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:

- Our port of the nhc98 runtime to Palm OS handhelds (→ 3.1.1) is running simple programs.

- Kate Stefanov is nearing the end of her project that applies off-the-shelf compression technology to reducing the size of interpreted programs (including functional ones).

- Matt Roberts has shifted focus and is now working on approaches to generic programming in lazy languages based on pattern calculus.

Further reading

Contact us via email to ⟨plrg@ics.mq.edu.au⟩ or find details on our 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 continuously extending the visual interactive programming environment Vital (→ 3.1.2) and Mark Callanan is working on type-sensitive editing operation in this context. Axel Simon maintains the gtk2hs binding to the Gtk+ GUI library (→ 4.5.3) in cooperation with Duncan Coutts, Oxford University. Chris Ryder is improving his Metrics and Visualization library Medina. Huiqing Li, Simon Thompson 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/
- 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, 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 (→ 3.2.1) and Parallel ML with skeletons (PMLS). Distributed/mobile languages include Glasgow distributed Haskell (→ 3.2.2), Erlang (http://www.erlang.org/), Hume (http://www-fp.dcs.st-and.ac.uk/hume/), JoCaml and Camelot.

Projects

Projects include

- High Level Techniques for Distributed Telecommunications Software 2002-06 is an EPSRC project (http://www.macs.hw.ac.uk/~dsg/telecoms/, GR/R88137) to evaluate high-level distributed programming techniques in a realistic telecommunications context.
- High Level Programming for Computational Grids 2003-05 is a 2-year British Council/DAAD funded travel grant (http://www.macs.hw.ac.uk/~dsg/projects/GpHGRID.html, Project No. 1223), with partners at LMU Munich, Phillips-Universitaet Marburg, and St Andrews University. We aim to evaluate a single large program on a computational grid, i.e., on a collection of grid-enabled workstation clusters.
- EmBounded Project (http://www.embounded.org/) 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 (http://www.macs.hw.ac.uk/~greg/SEAS/) 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 the C–>Haskell (→ 5.1.2) interface generator and more recently the hs-plugins (→ 4.2.12) library for dynamically loaded type-safe plugins. Based on runtime code (re)loading, we introduced a radically dynamic form of applications at the 2005 Haskell Workshop. These applications can hot swap their entire code base without losing application state. Following our new application architecture, we have implemented a highly customisable editor in Haskell, called Yi (→ 6.13), and also refactored the IRC bot lambdabot (→ 6.11).

In cooperation with Microsoft Research, Cambridge, we recently proposed *associated types* for Haskell type classes (see our papers at POPL 2005 and ICFP 2005). These are data type and synonym declaration in type classes that facilitate generic programming and may be used instead of functional dependencies. We are currently working at adding associated types to the Glasgow Haskell Compiler.

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

7.3.9 Logic and Formal Methods group at the Informatics Department of the University of Minho, Braga, Portugal

Report by: Jorge Sousa Pinto

We are a group of about 12 staff members and various PhD and MSc students. We have shared interest in formal methods and their application in areas such as data and code reverse and re-engineering, program understanding, and communication protocols. Haskell is our common language for teaching and research.

Haskell is used as first language in our graduate computers science education (→ 7.2.1). José Valença and José Barros are the authors of the first (and only) Portuguese book about Haskell, entitled "Fundamentos da Computação" (ISBN 972-674-318-4). Alcino Cunha has developed the Pointless library for pointfree programming in Haskell, as well as the DrHylo tool that transforms functions using explicit recursion into hylomorphisms. Supervised by José Nuno Oliveira, students Tiago Alves and Paulo Silva are developing the VooDooM tool (→ 5.3.4), which transforms VDM datatype specifications into SQL datamodels and students João Ferreira and José Proença will soon start developing CPrelude.hs, a formal specification modelling tool generating Haskell from VDM-SL and CAMILA. João Saraiva is responsible for the implementation of the attribute system LRC (→ 5.2.6), which generates (circular) Haskell programs. He is also the author of the HaLex library and tool, which supports lexical analysis with Haskell. Joost Visser has developed Sdf2Haskell (→ 5.2.7), which generates GLR parsing and customizable pretty-printing support from SDF grammars, and which is distributed as part of the Strafunski bundle. Most tools and library modules developed by the group are organized in a single infrastructure, to facilitate reuse, which can be obtained as a single distribution under the name UMinho Haskell Libraries and Tools.

The group is involved in the 3-year project called PURe which aims to apply formal methods to Program Understanding and Reverse Engineering. Haskell is used as implementation language, and various subprojects have been initiated, including Generic Program Slicing.

Further reading

LMF group home page (http://www.di.uminho.pt/~glmf) and PURe project home page (http://www.di.uminho.pt/pure). Version 1.0 of the UMinho Haskell Libraries and Tools has been released on April 5, 2005, and is available from http://wiki.di.uminho.pt/wiki/bin/view/PURe/PUReSoftware.

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 (→ 1.3): 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 darcs (→ 6.6), ghc (→ 2.1) and haddock (→ 5.5.9) have now been included in Fedora Extras! I hope to have more Haskell packages submitted and accepted in Extras soon. 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/ http://fedoraproject.org/wiki/Extras

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 (→ 5.5.9) and darcs (→ 6.6).

Additionally, both the stable and head branches of GHC are built nightly.

7.4.4 Haskell in Gentoo Linux

Report by: Andres Löh

The Gentoo Haskell team currently consists of Luis Araujo, Duncan Coutts, and Andres Löh. We get a lot of help from Lennart Kolmodin and Henning Günther.

We now have a Cabal (→ 4.1.1) eclass that allows us to write ebuilds for cabalized packages that consist of nearly no instructions and are therefore very easy to maintain.

There is ongoing work on a tool to convert Cabal package descriptions into ebuilds automatically, and to interface to Hackage.

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

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:

Monadic regions: the RGN monad

This is a type-class based implementation of the monadic regions described in Matthew Fluet and J. Gregory Morrisett's ICFP'04 paper. Region is a memory allocation technique introduced by Tofte and Talpin and implemented in ML-Kit and Cyclone. A region is an area of memory holding heap allocated data (reference cells). Regions may nest and so more than one region may be active at any given point. A new reference cell may only be allocated in an active region, and may then only be used while that region is active. The system statically guarantees that no cell can be accessed when its region is closed. The `RGN` monad, which is a variation of the `ST` monad, provides these static guarantees.

The code gives the first example of the *total* type comparison predicate, which can handle even non-ground types and quantified type variables.

http://pobox.com/~oleg/ftp/Haskell/types.html#monadic-regions

How to prove monad laws

The article http://pobox.com/~oleg/ftp/Computation/monads.html#proving-monad-laws demonstrates proving associativity of `bind`, on an example of a particular Error monad. The 'star' notation and a variable-free formulation of the associativity law turned out quite helpful.

How to make a Haskell function strict without changing its body

We show a simple trick of making a function strict in all or some of its arguments. We merely prepend one clause to the definition of the function, without making any changes to the other clauses. We illustrate the trick on an example of a Runge-Kutta iteration.

http://pobox.com/~oleg/ftp/Haskell/index.html#making-function-strict

7.5.2 Graham Klyne

Report by: Graham Klyne

My primary interest is in RDF http://www.w3.org/RDF/ and Semantic Web http://www.w3.org/2001/sw/ technologies. Since my submission for the November 2004 HC&A Report, I've been working at the Image Bioinformatics Research Group at Oxford University (http://www.bioimage.org/), and my plans to develop Swish, XML, RDF and description logic reasoning tools have somewhat taken a back seat (but have not been abandoned).

I have been using Haskell internally to process bioinformatics data, cross-referencing experimental result data with information in external databases of genetic information. It's all pretty trivial stuff so far, but I have hopes of demonstrating that functional languages can be a viable alternative to Excel spreadsheets for handling experimental data.

7.5.3 Inductive Inference

Report by: Lloyd Allison

Inductive Inference, i.e. 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), classification- (decision-) trees (supervised classification), Bayesian/causal networks/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 one day be embodied in a useful Haskell library or prelude for artificial-intelligence / data-mining / inductive-inference / machine-learning / statistical-inference.

A JFP paper (see below) appeared in January 2005, describing an early version of the software. Currently there are types and classes for models (various probability distributions), function models (including regressions), time-series (including Markov models), mixture models, and classification 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. A paper that includes the Bayesian network case-study is 'to appear'.

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

Future plans

Try to find a good name for this kind of programming: 'function' is to 'functional programming' as 'statistical model' is to what? 'Inductive programming' is the best name suggested so far – by Charles Twardy.

External factors slowed progress in 2005 but I hope to pick things up again soon. I want to develop time-series models further and must also look at template-Haskell, or similar, for dealing with csv-files in a nice way.

Further reading

- L. Allison. Models for machine learning and data mining in functional programming. J. Functional Programming, 15(1), pages 15–32, January 2005. doi: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 1.5b), 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. I'm currently developing a tool for automatic repeat detection in EST data.

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