Haskell Communities and Activities Report

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

Eighth Edition – May 13, 2005

Andres Löh (ed.)

Perry Alexander, Lloyd Allison, Tiago Miguel Laureano Alves
Krasimir Angelov, Alistair Bayley, Jérémy Bobbio
Björn Bringert, Niklas Broberg, Paul Callaghan
Mark Carroll, Manuel Chakravarty, Olaf Chitil
Koen Claessen, Catarina Coquand, Duncan Coutts
Philippa Cowderoy, Alain Crémieux, Iavor Diatchki
Atze Dijkstra, Shae Erisson, Sander Evers
Markus Forsberg, Simon Foster, Leif Frenzel
André Furtado, John Goerzen, Murray Gross
Walter Gussmann, Jurriaan Hage, Sven Moritz Hallberg
Thomas Hallgren, Keith Hanna, Bastiaan Heeren
Anders Höckersten, John Hughes, Graham Hutton
Patrik Jansson, Johan Jeuring, Paul Johnson
Isaac Jones, Oleg Kiselyov, Graham Klyne
Daan Leijen, Huiqing Li, Andres Löh
Rita Loogen, Salvador Lucas, Christoph Lüth
Ketil Z. Malde, Christian Maeder, Simon Marlow
Conor McBride, John Meacham, Serge Mechveliani
Neil Mitchell, William Garret Mitchener, Andy Moran
Matthew Naylor, Rickard Nilsson, Jan Henry Nyström
Sven Panne, Ross Paterson, Jens Petersen
John Peterson, Simon Peyton-Jones, Jorge Sousa Pinto
Bernie Pope, Claus Reinke, Frank Rosemeier
David Roundy, George Russell, Chris Ryder
David Sabel, Uwe Schmidt, Martijn Schrage
Peter Simons, Anthony Sloane, Dominic Steinitz
Donald Bruce Stewart, Martin Sulzmann, Autrijus Tang
Henning Thielemann, Peter Thiemann, Simon Thompson
Phil Trinder, Arjan van IJzendoorn, Tuomo Valkonen
Eelco Visser, Joost Visser, Malcolm Wallace
Ashley Yakeley, Jory van Zessen, Bulat Ziganshin

Preface

You are reading the 8th edition of the Haskell Communities and Activities Report (HCAR). These are interesting times to be a Haskell enthusiast.

Everyone seems to be talking about darcs (→ 6.3) and Pugs (→ 6.1) these days, and it is nice to see Haskell being mentioned in places where it usually was not. But here is what I think this new success really means: All the people who have spent their time experimenting with Haskell, writing tools or improving libraries, have done their job right! They have successfully lowered the barrier for newcomers. So thanks to *all* the contributors for their time and effort. All the entries, old and new, show that a lot is going on in the Haskell community, and that it is a lively and friendly place to be. The Haskell Cabal (→ 4.1.1), which is now finding its way into all the major Haskell implementations, will hopefully make the contribution of Haskell libraries and tools even easier from now on.

A special thank-you also goes to all the people who help spreading the Haskell word in new ways: the Monad.Reader (→ 1.5) is a new Haskell on-line magazine that already produced some very informative and well-written articles, the Haskell Sequence (→ 1.4) is a new site for news and discussion. I also want to mention Walter Gussmann's entry here (→ 7.2.2), who has a success story to tell about teaching functional programming to children in high school.

Because the HCAR is quickly increasing in size, I have removed a couple of entries from authors that have not reported back. If the projects are still active, I will be more than happy to include them again in the next edition. I kept the typographical hints indicating change: completely new entries have a blue (or gray, if viewed without color) background; entries with a certain amount of change have a header with a blue background. I have also slightly adapted the structure of the report. Feedback is, as always, welcome ⟨hcar@haskell.org⟩.

Please remember that the next report will appear in November 2005, so already mark *the last weeks of October*, because the new entries will be due by then.

Editing the report has been an enjoyable experience, and I sincerely hope that you will enjoy reading it even more.

Andres Löh, University of Utrecht, The Netherlands

Contents

1 General

1.1 haskell.org

Report by: John Peterson

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 John Peterson ⟨peterson-john@cs.yale.edu⟩ 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.

Thanks to Fritz Ruehr for making the cafepress store on haskell.org a lot more exciting and to Jonathan Lingard for adding some nice style sheets to our pages.

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.

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 cvs repository so that we can more easily share out maintenance.
- add searching capability to haskell.org.

Some of these ideas would be good student projects. Be lazy – get students to do your work for you.

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. 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, 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!

1.4 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

I am currently in the final-stages of producing an introductory Haskell textbook. The book is a revised and extended version of my Haskell course at the University of Nottingham, which has been developed and class tested over many years. The first seven chapters (97 pages) are available for preview on the web: http://www.cs.nott.ac.uk/~gmh/book.html

I'd be pleased to make the full current draft (162 pages) available to anyone that is teaching Haskell and may be interested in using the book in their course; please contact me for further details.

1.6.2 Haskell Tutorial WikiBook

Report by: Paul Johnson

I recently 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

1.7 Haskell related events

1.7.1 Future events

You may want to participate in some of the following Haskell-related events:

ICFP 2005 The 10th ACM SIGPLAN International Conference on Functional Programming, this year in

Tallinn from September 26 to 28. See http://www.brics.dk/~danvy/icfp05/.

TFP 2005 The Trends of Functional Programming is held this year for the sixth time – guess where – in Tallinn, from September 23 to 24. See http://www.tifp.org/tfp05/.

HW 2005 The Haskell Workshop, as always co-located with ICFP, and therefore in Tallinn, on September 30. See http://www.cs.uu.nl/~daan/hw2005/.

CUFP 2005 The will be another Commercial Users of Functional Programming meeting co-located with ICFP – more information will be available soon.

If you would like to see other relevant events mentioned in this section, please submit pointers for the next edition of the HC&A Report.

2 Implementations

2.1 The Glasgow Haskell Compiler

Report by: Simon Peyton-Jones

Despite being "full", GHC continues to develop at an alarming rate. Here are some highlights from the last few months.

- We finally released GHC 6.4, over a year after the last major release, in March 2005. The long gap between releases is partly because doing a full release is a lot of work. But a bigger reason is that a lot is still happening to GHC, and it can be hard to find a moment when all the dust has settled at once.

- With major input from Tim Harris, we implemented Software Transactional Memory in the autumn, a new synchronisation and communication for Concurrent Haskell that completely replaces `MVars`. (`MVars` are still available, but they become library.) We think STM is a major leap forward, and are eager to hear your experiences when you try it out. Anyone who uses Concurrent Haskell should give it a whirl. Paper at http://research.microsoft.com/~simonpj/papers/stm.

- Much better support for mutually-recursive modules. Mutual recursion used to be handled by hand-written, but completely unchecked,interface files. Now you still need to write a loop-breaking module interface by hand, but it's regarded as a source file, in Haskell syntax; it is fully typechecked; it is understood by `ghc --make`; it is checked against its parent source file when the latter is compiled; and so on. All vastly superior. Further improvements (e.g. ability to include instance declarations) are in the works.

- Much improved support for Cabal (→ 4.1.1).

- A declaration type signature now binds lexically-scoped type variables. For example

  ```
  tail :: forall a. [a] -> [a]
  tail (x:xs) = xs :: [a]
  ```

  Here, the `a` bound by the `forall` scopes over the body of `tail`. I fought against this for ages, but it's just so convenient. It only happens if you give an explicit `forall`, though.

- Improvements in Template Haskell support; notably the of an abstract type `Name` for the names of TH variables, rather than `String`, and ramifications thereof (http://www.haskell.org/ghc/docs/papers/th2.ps).

- Modest improvements to the implementation of Generalised Algebraic Data Types (GADTs). The big shortcoming is that GADTs do not interact nicely with type classes and functional dependencies yet. That turned out to be a more complex task than we'd anticipated.

- Rebindable syntax now works properly for do-notation. The idea is that, with `-fno-implicit-prelude`, do-notation type-checks and desugars just as if you had written the program with explicit `(>>=)` and `(>>)` etc, using whatever `(>>=)` operator is in scope in this module. There's no longer a requirement that the type looks exactly like that for the built-in `(>>=)` operator.

- Support for the amd64 (x86-64) platform is improving. The 6.4 release had basic registerised support for this processor; we have now added a fully-working native code generator and completed support for the FFI. GHCi support should be forthcoming shortly. We aim for this to be a fully-supported platform in future GHC releases.

- John Goerzen is experimenting with converting the CVS repository to darcs (→ 6.3). We hope to set up a two-way sychronisation between the two repositories, so that people working on GHC can use whichever source control system they prefer (though we hope the general trend will be in the direction of darcs, so that eventually we can drop CVS support).

- We are working on adding an API to GHC, so that you can use GHC itself as a Haskell library (by saying "`import GHC`"). This will let you typecheck, compile, and execute all of GHC-supported Haskell from your own application. GHC's own front-ends are implemented on top of this API: GHCi, `--make` and the command-line interface. The new Visual Studio mode is also using this API to support interactive type checking of source code as it is edited (amongst many other things).

  For the current version of the API, see http://cvs.haskell.org/cgi-bin/cvsweb.cgi/~checkout~/fptools/ghc/compile r/main/GHC.hs?rev=1.22.

  We plan to provide the GHC API via a package (`-package ghc`) in the next major release of GHC (6.6).

The Big New Thing over the next few months will be multi-processor GHC. Now that multi-core processors are on the near horizon, and STM has given us a nice way to coordinate them, we are building a multi-processor GHC that uses multiple processors running Haskell on a single shared heap. That involves fine-grain locking on thunks, but we have a way to make that quite cheap. This is quite complementary to the long-standing work on GUM, aimed at more distributed-memory architectures with disjoint heaps.

On the type system front, we plan to extend GHC's higher-rank type system to incorporate impredicative types too: http://research.microsoft.com/~simonpj/papers/boxy.

Thank you to everyone who completed the GHC survey. If you use GHC and have not completed the survey, please do so – we are keen to get an unbiased sample of our actual users, rather than one skewed towards hard-core Haskell devotees. Here it is: http://www.haskell.org/ghc/survey/start.cgi. We'll publish the results once we've digested them.

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

An interim release of Hugs appeared in March 2005, on the same day as releases of GHC and nhc98. This release was mainly targeted at Unix users (see below). It featured Unicode support (contributed by Dmitry Golubovsky) and lots of up-to-date libraries. Additions include the graphics library used in the "School of Expression" textbook.

A major new feature is support for the Cabal infrastructure (→ 4.1.1), which is now the recommended way to install third-party packages with Hugs. Indeed the Hugs build system uses Cabal to prepare all the libraries included with Hugs (after a little bootstrapping).

Sadly no-one is packaging Hugs for Windows. It will probably build under MinGW or Cygwin with little or no work, but no-one has tried it recently. The new Cabal-based library build system holds out the promise of independence from the Unix-like environments, but Cabal itself needs more work under Windows. 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 was recently refreshed to version 1.18. Maintenance continues in CVS at haskell.org.

Tony Sloane has recently added a literate version of nhc98's runtime system kernel to CVS. We hope this will enable more people to understand the internals, and more easily contribute to the compiler. Some individual projects of potential interest to many users would be:

- overhaul the type inference subsystem, towards the goal of implementing multi-parameter classes;
- add pattern-guards to the source language;
- remove any 32-bit platform assumptions;
- add concurrent threads using a co-operative scheduler;
- implement exceptions.

Further reading

http://haskell.org/nhc98

2.4 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 (→ 3.2) 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.

2.5 Haskell to Clean Translation

Report by:	Matthew Naylor

The primary aim of the project is to develop a tool, which we name Hacle, for translating Haskell programs to Clean programs, thereby allowing the Clean compiler to compile Haskell programs. The question is, *can the Clean compiler, in combination with Hacle, produce faster executables than existing Haskell compilers?*

The answer, perhaps rather predictably, is *sometimes yes*. We have noticed that, in some cases, the hybrid Hacle-then-Clean compilation system can produce executables which are up to a factor of four times faster than the corresponding GHC-compiled programs. However, we suspect that these cases are in a minority. Nevertheless, to be of any significance at all, we must also argue Hacle's completeness.

Hacle can translate programs which conform to a slightly restricted Haskell 98 standard. It can translate itself, which is written in approximately fifteen thousand lines of code and makes use of many of the features provided by Haskell 98. This result positively demonstrates reasonable completeness.

The project is effectively finished; this is not to say that the tool cannot be improved, rather that we are content with its current state. Only the unlikely event of widespread use would motivate such improvements. However, the following question is unanswered: why do Clean and GHC sometimes outperform each other?

For more information including detailed technical documentation, my dissertation, more results, Hacle's limitations, and a download link to Hacle, see the project's web page.

Further reading

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

Grateful acknowledgements to Malcolm Wallace and Olaf Chitil.

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 (→ 3.4.4) (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. We are focussing our current attention on reworking the nhc98 runtime kernel by writing a literate version to make it easier to understand and port. We hope to use this work as the basis of a new Palm OS port that is more reliable and maintainable than our previous version.

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

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

3.1.3 hOp

Report by:	Jérémy Bobbio and Thomas Hallgren
Status:	beta, active development

hOp is a micro-kernel based on the run-time system (RTS) of the Glasgow Haskell Compiler. It is meant to enable people to experiment with writing various components of an operating system in Haskell. This includes device drivers, data storage devices, communication protocols and tools required to make use of these components.

The February 2004 release of hOp consisted of a trimmed-down RTS that does not depend on features usually provided by an operating system. It also contains low-level support code for hardware initialization. This release made most functions from the base hierarchical library available (all but the System modules), including support for threads, communication primitives, and the foreign function interface (→ 3.2).

Building on the features of the initial release, we designed and implemented an interrupt handling model. Each interrupt handler is run in its own thread, and sends events to device drivers through a communication channel. We tested our design by implementing a simple PS/2 keyboard driver, and a "shell" that allows running a "date" command, which accesses the real time clock of the computer. A release of hOp containing these additional features was made in June 2004.

Iavor Diatchki, Thomas Hallgren, and Andrew Tolmach made some additions to hOp. The resulting system is in an experimental state and is preliminary called House. The additions include a PS/2 mouse driver, using VBE 2.0 to setup a linear frame buffer for graphics, a window system implemented in Haskell (Gadgets, developed by Rob Noble and Colin Runciman at the University of York), new primitives for setting up demand paged virtual memory and executing arbitrary machine code in protected mode. The function that executes code in user mode returns when normal execution is interrupted for some reason (e.g., by a hardware interrupt, a system call or a page fault), allowing Haskell code can handle the situation in an appropriate way, and then resume user mode execution, if that is appropriate.

A recent addition to the system is a driver for NE2000 compatible network cards (as emulated by QEMU) and a simple protocol stack. We have used this to add shell commands for downloading files via TFTP, and then display them on the screen or execute them as user mode binaries.

Further reading

Further information, source code, demos and screenshots are available here:
- http://www.macs.hw.ac.uk/~sebc/hOp/
- http://www.cse.ogi.edu/~hallgren/House/

3.1.4 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 (→ 7.3.8). 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 (→ 4.2.11).

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 (→ 7.3.8).

3.1.5 Haskell Server Pages (HSP)

Report by:	Niklas Broberg
Status:	experimental, latest release: 0.2 (May -05)
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 (→ 4.2.11).
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

3.1.6 Haskell Regular Patterns (HaRP)

Report by:	Niklas Broberg
Status:	stable, currently not actively developed, latest release: 0.2 (April 05)
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 Foreign Function Interface

Report by:	Manuel Chakravarty
Status:	Version 1.0

The specification of the Haskell 98 Foreign Function Interface 1.0 is now also available in HTML. To download or browse online, please visit http://www.cse.unsw.edu.au/~chak/haskell/ffi/.

3.3 Non-sequential Programming

3.3.1 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.

System Evaluation and Enhancement

The first 3 items are linked by a British Council/DAAD collaborative project between Heriot-Watt University, St Andrews University, and Phillips Universität Marburg.

- We are adapting GpH to run on computational GRIDs. The current implementation performs well on single clusters, and multiple clusters with a low-latency interconnect. A distribution is available on request from ⟨ceeatia@macs.hw.ac.uk⟩.
- We are designing a generic parallel runtime environment encompassing both the Eden (→ 3.3.3) and GpH runtime environments
- 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).

Implementations

The GUM implementation of GpH is available in two development branches, and work on a port of GUM to the latest GHC 6.xx branch has been started over summer.

- The stable branch (GUM-4.06, based on GHC-4.06) is available for RedHat-based Linux machines: binary snapshot (see installation instructions). 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 working and has been used on a Beowulf cluster. It is available on request as a source bundle.

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

Further reading

GpH Home Page: http://www.macs.hw.ac.uk/~dsg/gph/

3.3.2 GdH – Glasgow Distributed Haskell & Mobile Haskell

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

Implementation:

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 (→ 3.3.1). A beta release of mHaskell will be available in December 2005.

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

3.3.3 Eden

Report by:	Rita Loogen

Description

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

in Madrid: Ricardo Peña, Yolanda Ortega-Mallén, Mercedes Hidalgo, Rafael Martínez, Clara Segura

in Marburg: Rita Loogen, Jost Berthold, Claudia Kerber, Steffen Priebe, Pablo Roldán Gómez

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 (see above). 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(4), 2005, to appear.

semantics Mercedes Hidalgo-Herrero, Alberto Verdejo, Yolanda Ortega-Mallén: *Looking for Eden through Maude and its strategies*, submitted, 2005.

analyses Clara Segura, Ricardo Peña: *Nondeterminism analyses in a parallel-functional language*, Journal of Functional Programming 15(1), pp. 67–100. January 2005.

compilation Steffen Priebe: *Preprocessing Eden with Template Haskell*, April 2005, submitted.

profiling Pablo Roldán Gómez, J. Berthold: *Eden Trace Viewer: A Tool to Visualize Parallel Functional Program Executions*, March 2005, submitted.

skeleton performance analysis Jost Berthold, Rita Loogen: *Improving Functional Topology Skeletons with Dynamic Channels*, March 2005, submitted.

Current Activities

- Yolanda and Mercedes are working on an implementation of Eden's operational semantics in Maude.
- Jost is working on a more general implementation of parallel Haskell dialects in a shared runtime system.
- Steffen continues his work on the polytypic skeleton library for Eden making use of the new metaprogramming facilities in GHC.
- Pablo is working on extensions of the Eden trace viewer tool.
- Jost and Rita continue working on the skeleton library.
- Claudia and Rita investigate parallel functional graph algorithms.

Further reading

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

3.3.4 HCPN – Haskell-Coloured Petri Nets

Report by:	Claus Reinke
Status:	slow progress

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.

I have just returned to this project, working on several items: first, the embedding of HCPN in Haskell has changed slightly. Apart from making the generated transition code even simpler, the idea is to abstract from the precise representation of places, in order to prepare the necessary move towards hierarchical HCPN (the original embedding mapped places directly to record fields, making composition of nets somewhat difficult without extensible records). Second, I am moving the drawing code from wxHaskell to HOpenGL (→ 4.6.1) – if HOpenGL's support is somewhat "basic" (you want higher-level abstractions on top), wxHaskell's drawing can only be described as "primitive" (encouraging bad habits, unless you avoid some of its features).

Due to all these ongoing rewrites, the current sources are not in a releasable state, but until this settles down, the old snapshots are still available from the project web page. 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/

3.4 Type System/Program Analysis

3.4.1 Agda: An Interactive Proof Editor

Report by:	Catarina Coquand
Status:	active development

Agda is an interactive type-based editor for editing proofs and programs that has been developed at Chalmers and Göteborg University. It builds on previous work at Chalmers such as ALF and Cayenne. It implements a proof/type checker for a language that is based on Martin-Löf Type Theory. We are experimenting with how such a proof language could be extended with data-types, modules and records. The syntax of the language is rather close to Haskell. The language can also be seen as a start for a dependently typed programming language.

The program is written in Haskell and it consists of roughly 15 000 lines of code. It is connected with one graphical and one text-based interface. The graphical interface Alfa http://www.cs.chalmers.se/~hallgren/Alfa/ is written in Haskell using Fudgets. The is also a "simple" emacs-interface which doesn't know the syntax of the language and communicates via a text-based protocol with Agda. This interface comes with the distribution of Agda.

Agda is running with a stable version that is slightly more than one year old. It is also possible to download newer unstable versions. In this new version experiments are done with hidden arguments as in Cayenne, addition of over-loading with a class system and built-in types such as characters, strings and integers.

We have recently started a collaboration with AIST (Advanced Industrial Science and Technology Institute in Japan) on development and applications of Agda. In particular on writing better documentation and integration with other automatic proof tools.

Agda source code can be browsed at http://cvs.coverproject.org/marcin/cgi/viewcvs/ and can be accessed by anonymous CVS from cvs.coverproject.org.

Short term goals are among many things:

- Write a better documentation of the code and the system.
- Examples of classes and built-in types
- Building on the libraries
- Revision of the type-checking algorithm
- Connecting Agsy with the emacs-interface – Agsy is an automatic proof search plugin for Alfa for the moment.

Further reading

For more details about the project, read about QuickCheck (→ 5.4.4) and Cover (→ 7.3.10) in this report or consult the homepage at http://www.cs.chalmers.se/~catarina/agda/.

3.4.2 Epigram

Report by:	Conor McBride

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

thanks to Thorsten Altenkirch, Epigram has a team of willing workers in Nottingham. A new implementation (also in Haskell) is in progress, incorporating a compiler based on Edwin Brady's doctoral research.

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.

Implementation

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. On the language side, considerable use is made of monad transformers, higher-kind polymorphism and type classes. Moreover, its denotational approach translates Epigram's lambda-calculus directly into Haskell's. On the tool side, Haskell's profiler (in the capable hands of Paul Callaghan) has proved invaluable for detecting bottlenecks in the code.

Current Status

Epigram can be found on the web at http://sneezy.cs.nott.ac.uk/epigram/ 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://sneezy.cs.nott.ac.uk/epilogue/ will be much less rudimentary.

3.4.3 Chameleon

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

Latest developments:

Chameleon implementation

We are working on a completely new implementation of the Chameleon compiler. Chameleon is a Haskell-like language which supports almost all of Haskell 98, as well as the following extensions (and more):

- Multi-parameter type classes, with functional dependencies
- Type-level constraint programming (using CHR)
- Lexically scoped type annotations (see below)
- Generalized guarded recursive data types (see below)

The new implementation incorporates a significantly faster constraint solver, and has been designed to be easily extended. In particular, language extensions which make use of the underlying solver can straightforwardly take advantage of the system's advanced error reporting features. We are currently working on a backend for the compiler, as well as the integration of many commonly available Haskell libraries.

Type system extensions

More general algebraic data types: We formalize an extension of Hindley/Milner with a user-programmable constraint domain and a general form of algebraic data types (GRDT) which unifies common forms such as existential types, the combination of type classes and existential types and the more recent extension of guarded recursive data types. We also support advanced type class extensions such as functional dependencies. Thus, we can express novel variants which allow for a GRDT-style behavior of type classes with existential types

Lexically scoped type annotations: We generalize the two common forms of type-sharing (found in GHC) and type-lambda annotations (found in ML) leading to an expressive system of lexically scoped annotation. We show that such an extension is highly useful in case of advanced typing features such as polymorphic recursion, type classes and guarded recursive data types.

A fresh look at kind inference and kind checking: We present an improved error reporting scheme for kind inference and checking based on our earlier work on type error reporting. We consider a monomorphic kind system. Hence, polymorphic kinds resulting from kind inference will be replaced by monomorphic kinds. This is known as defaulting of inferred kinds. The standard approach is to default such polymorphic kinds to $*$ (the kind of types). The problem is that failure of kind checking may be due to kind defaulting. Hence, we introduce a novel kind validation system that first performs kind checking to determine the most general kind environment. Then, we test that the actual inferred kinds agree with this kind environment. Our approach represents a nice application of the principal kinding property of monomorphic kind languages.

Language extensions

XHaskell – adding regular expression types and pattern matching to Haskell: Our overall goal is to add XDuce-style features to full Haskell.

In a first step we consider a type-driven translation from XDuce to Haskell (standard Hindley/Milner fragment) based on a structured representation of XDuce values. XDuce type inference guides the insertion of appropriate coercion functions such that the resulting Haskell program is type correct and reflects the meaning of the original XDuce program.

In an actual implementation, we plan to make use of the regular expression library mentioned below.

Regular expression library

We introduce a novel implementation of subtyping among regular expression types in terms of Haskell-style type classes by making use of two type class extensions. We require overlapping and co-inductive instances to encode a proof system to decide subtyping among regular expressions. We assume that each regular expression type has some underlying structured runtime representation. Hence, we not only check for the containment problem among regular expressions, but also automatically derive some appropriate casting functions among the underlying structured values.

Further reading

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

3.4.4 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.

Since the report of November 2004

- Bastiaan Heeren has finished his PhD thesis, which has been accepted by the promotion committee. It can be downloaded from http://www.cs.uu.nl/people/bastiaan/phdthesis/.
- A paper was published in PADL 2005, discussing type inference directives for Haskell 98 type classes.
- A student is working on analyzing the loggings of Helium to obtain information about how students program, how 'effective' our hints are, and so on.
- A CD with the clean logged programs for two functional programming courses is available. If you have a need for such a CD, contact us, and we can discuss this.
- A second student is working on improving the hint facility for Helium, which generalizes the siblings and permutation facility.
- Other current work involves generating type inferencers.

Further reading

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

3.4.5 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 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 is used as well as other formalisms allowing compact notation like parser combinators.

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 these features has been described at the AFP 2004 summerschool (lecture notes yet to appear, handouts are available).

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

Our plans for the near future are to complete the description of all steps.

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
- AFP handouts: http://www.cs.uu.nl/research/techreps/UU-CS-2004-037.html
- 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/

3.5 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 2.1.1 from April 2005.

Languages

Light-weight generic programming There are a number of approaches to light-weight generic programming. The latest contributions are from the 'Scrap your boilerplate' approach.

Generic functions for data type traversals can (almost) be written in Haskell itself, 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 '04) paper "Scrap more boilerplate: reflection, zips, and generalised casts" and an unpublished paper "Scrap your boilerplate with class: extensible generic functions". The former paper shows how to fill some of the gaps (such as generic zips) which previously were difficult to solve in this approach. The latter paper shows how you can turn 'closed' definitions of generic functions (not extensible when new data types are defined) into 'open', extensible, definitions.

Until now, there have been applications for which Hinze and Peyton Jones's "Derivable type classes" would work, but SYB-style generic programming would not. The latest SYB paper shows how SYB-style programming can handle this class of applications too, so Simon is planning to remove derivable type classes from GHC (Section 7.11 of the GHC 6.4 user manual). Please let him know if that would be a problematic for you.

In "Generic proofs for combinator-based generic programs" (TFP 2004), Fermin Reig shows how to write generic proofs for generic programs that use the SYB library. The idea is that generic functions implemented using type classes can also be expressed in Generic Haskell, and this allows us to write more concise proofs.

Generic Haskell Andres Löh successfully defended his PhD thesis "Exploring Generic Haskell" on September 2, 2004. The thesis describes Dependency-style Generic Haskell, and introduces, amongst others, a new type system for Generic Haskell that at the same time simplifies the syntax and provides greater expressive power. Electronic copies are available at http://www.cs.uu.nl/~andres/ExploringGH.pdf.

The Coral release of the Generic Haskell compiler (January 2005) implements Dependency-Style Generic Haskell.

Generic Haskell is used in "Generic validation in an XPath-Haskell data binding" by Rui Guerra, Johan Jeuring, and Doaitse Swierstra, Plan-X 2005, to implement a typed Haskell-XPath data binding. Furthermore, Stefan Holdermans, Johan Jeuring and Andres Löh show how to add 'views' to Generic Haskell in "Generic views on data types" (http://www.cs.uu.nl/research/techreps/UU-CS-2005-012.html).

Current Hot Topics

Generic Haskell: inferring types of generic functions; finding transformations between data types. Other: the relation between generic programming and dependently typed programming; the relation between coherence and generic programming; better partial evaluation of generic functions; methods for constructing generic programs.

Major Goals

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, NHC98, and Hugs 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`, 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.

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 (→ 4.7.5) interface. The prototype client, cabal-get, can download and install a package and its dependencies.

Further reading

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

4.1.2 Eternal Compatibility in Theory – a module versioning protocol

Report by: Sven Moritz Hallberg

I've recently 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 (→ 4.2.2) 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

4.1.3 LicensedPreludeExts

Report by:	Shae Erisson

The PreludeExts wiki page started with an oft-pasted email on the `#haskell` IRC channel, 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 General libraries

4.2.1 Process

Report by:	Bulat Ziganshin
Status:	beta, actively developed

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

4.2.2 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/

4.2.3 TimeLib

Report by:	Ashley Yakeley
Status:	active development

TimeLib is my informal name for an effort to redesign the current library for handling Time (System.Time), picking up from Simon Marlow's earlier effort. A long discussion on the libraries list in January and February hashed out some of the essential ideas to be represented and some of the design fundamentals, and I have now started implementation.
There's a darcs repository if you want to follow along at home, but currently much of the code is fairly tentative and tends to change rapidly as I try to seek a balance between expressive functionality and intelligible simplicity. When the code becomes more stable I will seek comment from the community.

Further reading

http://semantic.org/TimeLib/

4.2.4 A redesigned IO library

Report by:	Simon Marlow

Some time ago on the libraries mailing list there was a discussion about a replacement for Haskell's IO library. The main aims are:

- To separate underlying IO objects (files, pipes, sockets etc.), from a general notion of Streams, providing improved
  1. Type Safety: certain operations only make sense for certain kinds of IO objects. For example `hFileSize` only makes sense on files, not sockets. Also, input streams would be separate from output streams.
  2. Generality: Under this scheme, programmers would be able to implement their own Streams (something which cannot be done with Handles).
- To allow translations to be layered on top of Streams in a general way. The most common type of translation is a text encoding, which translates between the external encoded form of text (say, UTF-8) and Haskell's Unicode Char type. This addresses a serious deficiency in Haskell's current IO library, namely the lack of support for specifying a character translation.
- More features: e.g. mapped file support.

See the libraries archives for the discussion, e.g.

- http://haskell.org/pipermail/libraries/2003-July/001298.html
- http://haskell.org/pipermail/libraries/2003-July/001299.html
- http://haskell.org/pipermail/libraries/2003-August/001313.html

Since the previous report some progress has been made on a prototype, which is available here: http://haskell.org/~simonmar/new-io.tar.gz.

The prototype currently supports only basic I/O using files, but has some support for internationalization. I (Simon M.) am not actively working on this at the moment, so anyone that would like to pick this up is entirely welcome.

4.2.5 The Haskell Cryptographic Library

Report by:	Dominic Steinitz

The current release is 2.0.1. New, since the last report, is a complete re-write of the ASN.1 handling modules, the ability to handle keys stored X.509 certificates, the inclusion of Codec.Binary.Base64, lots of tests using HUnit and QuickCheck (→ 5.4.4) and the use of a darcs (→ 6.3) repository all packaged using Cabal (→ 4.1.1).

The library now supports: DES, Blowfish, AES, Cipher Block Chaining (CBC) mode, PKCS5 and nulls padding, MD5, SHA-1, Base64, RSA, OAEP, ASN.1, PKCS#8 and X.509.

The library follows the hierarchical standards and has Haddock (→ 5.5.6) style documentation. There are demo / test programs using published test vectors and instructions on how to use RSA in Haskell and interwork with openssl. In particular, you can generate key pairs using your favorite method (openssl, for example) and then use them in Haskell. Not only can you now read a private key into your Haskell program via PKCS#8 and use it to decrypt something encrypted with your public key but you can also read a public

key into your Haskell program via X.509 and use it to encrypt something for decryption using your private key.

There is still plenty of existing code that should be incorporated such as RC4 (courtesy of Doug Hoyte). With the new ASN.1 handling it should be straightforward to add a PKCS#12 module. The next piece of work is likely to be support for digital signatures.

All contributions are welcome.

Further reading

http://www.haskell.org/crypto

4.2.6 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.

Further reading

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

4.2.7 Haskore revision

Report by:	Henning Thielemann
Status:	experimental, active development

Haskore is a set of Haskell modules by Paul Hudak that allow music composition within Haskell, i.e. without the need of a custom music programming language. In general this project aims at improving consistency throughout the package, revising design decisions, fixing bugs, and eventually extending Haskore. In particular some improvements are: The Music structure is based on a more general temporal media data structure as proposed by Paul Hudak. The core Music data structure is hidden by functions that work on it. The support for infinite Music objects is improved. You can feed CSound with infinite music data through a pipe and you can feed an audio file player like Sox with an audio stream entirely rendered in Haskell (see Audio Signal Processing project (→ 6.16)) The test suite is now based on QuickCheck (→ 5.4.4) and HUnit. The AutoTrack project is adapted and included now.

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/

4.2.8 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 transformer library now has its first official release. I have put it on my web page: http://www.cse.ogi.edu/~diatchki/monadLib

It is in many ways similar to what's distributed with GHC/Hugs/etc, but I think also simplified and better organized. The library interface is documented with haddock (→ 5.5.6). The monads/transformers currently in the library are:

- ReaderT (environment)
- WriterT (output)

- StateT
- ExceptT
- BackT
- ContT

In this version I decided to implement some of the transformers (backtracking,exceptions) in continuation passing style, thinking that they may work better that way. I haven't done any formal testing on that though. The Haskell extensions the library uses are:

- Multiparameter classes (important).
- Rank-2 polymorphism (for the CPS implementations, could be removed).
- Functional dependencies (could be removed, but is likely to require more type annotations).

For any questions, comments, or bug reports please send me a mail.

Further reading

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

4.2.9 HBase

Report by:	Ashley Yakeley
Status:	stalled

HBase is a large collection of library code, compiled "`-fno-implicit-prelude`", intended as an experimental/alternative reorganized interface to the existing standard libraries making full use of GHC's extensions. HBase development is driven by HScheme (→ 6.2) and my other Haskell projects, and sometimes by whatever interests occur to me. Right now it includes:

- a library of various classes of Functors and Monads,
- transformation, encoding and property functions for Unicode,
- types and classes for parsing,
- functions for parsing XML and RDF,
- code for constructing SQL queries,

. . . and much else. I'm hoping some of the ideas might eventually make their way into standard libraries, or perhaps the standard libraries of some future extended "Haskell 2".

Very little work is currently being done on it.

Further reading

http://sourceforge.net/projects/hbase/

4.2.10 Pointless Haskell

Report by:	Jorge Sousa Pinto

Pointless Haskell is a library for point-free programming with recursion patterns defined as hylomorphisms. It is part of the UMinho Haskell libraries that are being developed at the University of Minho (→ 7.3.8). The core of the library is described in "Point-free Programming with Hylomorphisms" by Alcino Cunha.

Pointless Haskell also allows the visualization of the intermediate data structure of the hylomorphisms with GHood. This feature together with the DrHylo (→ 5.2.8) tool allows us to easily visualize recursion trees of Haskell functions, as described in "Automatic Visualization of Recursion Trees: a Case Study on Generic Programming" (Alcino Cunha, In volume 86.3 of ENTCS: Selected papers of the 12th International Workshop on Functional and (Constraint) Logic Programming. 2003).

Further reading

The Pointless Haskell library is available from http://wiki.di.uminho.pt/bin/view/Alcino/PointlessHaskell.

4.2.11 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 standard 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 0.9.8 has been released.

Further reading

Source and documentation can be found at http://www.cse.unsw.edu.au/~dons/hs-plugins.

4.2.12 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

4.2.13 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 (→ 4.2.12)). 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 Parsec

Report by:	Daan Leijen
Status:	stable

Parsec is a practical parser combinator library for Haskell that is well documented, has extensive libraries, and good error messages. It is currently part of the standard Haskell libraries (in `Text.ParserCombinators.Parsec`) and has been stable for years now. We plan to add a module that adds combinators to parse according to the (full) Haskell layout rule (available on request).

Further reading

http://www.cs.uu.nl/~daan/parsec.html

4.3.2 Haskell-Source with eXtensions (HSX, haskell-src-exts)

Report by:	Niklas Broberg
Status:	beta, maintained, latest release: 0.2 (April 05)

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 (→ 3.1.6) extension as well as HSP-style embedded XML syntax (→ 3.1.5).

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 October 2004
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 (→ 3.5);
- the library ATermLib for data exchange;
- the tool Sdf2Haskell (→ 5.2.5) for external parser and pretty-print integration.

The Strafunski-style of generic programming can be seen as a lightweight variant of generic programming (→ 3.5) because no language extension is in-

volved, 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" (→ 3.5) 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 (→ 5.2.7), 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 (→ 7.3.8) 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/

4.3.4 Medina – Metrics for Haskell

Report by:	Chris Ryder

The Medina library is a Haskell library for GHC that provides tools and abstractions with which to build software metrics for Haskell programs.

The library includes a parser and several abstract representations of the parse trees and some visualization systems including pretty printers, HTML generation and callgraph browsing. The library has some integration with CVS to allow temporal operations such as measuring a metric value over time. This is linked with some simple visualization mechanisms to allow exploring such temporal data. These visualization systems will be expanded in the near future.

We have carried out case studies to provide some validation of metrics by looking at the change history of a program and how various metric values evolve in relation to those changes. In order to do this we implemented several metrics using the library, which has given some valuable ideas for improvements to the library.

Following on from the case studies we have improved and extended the visualization systems and implemented some of the ideas from the case studies. Demos and screenshots are available on the Medina webpage: http://www.cs.kent.ac.uk/~cr24/medina.

Currently there is no released version of the Medina library, but my PhD thesis has been submitted so I am now in the process of preparing a release. This should be available real-soon-now.

4.4 Data handling

4.4.1 DData

Report by:	Daan Leijen
Status:	stable

DData is a library of efficient data structures and algorithms for Haskell (Set, Bag, and Map). It is actively maintained and stable.

DData is currently included in the standard hierarchical module name space and ships with GHC, NHC, and Hugs. This will be the last entry in the communities report.

Further reading

http://www.cs.uu.nl/~daan/ddata.html

4.4.2 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. The HList library relies on common extensions of Haskell 98.

We added two general HList (to be more precise, HRecord) functions inspired by OOHaskell: HLeftUnion and Narrow. The latter narrows a record to a different record type. We included two new examples: Joy, with the *typed* stack, in Haskell; lists of heterogenous "objects" implementing the same interface, without the use of existentials. We also made a few slight changes to make the library work with GHC 6.4.

Further reading

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

4.4.3 HSQL

Report by:	Krasimir Angelov
Status:	stable

The HSQL is a simple library for database access from Haskell. It is relatively small and complete. bug fixes are always welcome and If someone is wishing to add a new backend I will be glad to help him.

Further reading

http://htoolkit.sourceforge.net/

4.4.4 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 (→ 4.4.3), 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.

Since the last report we've added support for bind variables. The plan to redesign the interface to use just the IO monad didn't work out well, so we've abandoned that for now and have retained the existing monad-transformer-based design. Oleg has done some impressive refactoring/simplification work, which has better separated the enumerator (front-end) library from the various implementations (back-ends). The result is that it's much easier to implement new back-ends, and the existing back-ends are much simpler.

Alain Cremieux is attempting a BerkeleyDB back-end (our first non-SQL one), Alistair has started on an MS Sql Server back-end, and Oleg plans to do a PostgreSql one.

Further reading

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

4.4.5 HaskellDB

Report by:	Anders Höckersten
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 (→ 4.4.3) or wxHaskell (→ 4.5.1). HaskellDB was originally developed by Daan Leijen. Development was restarted as part of a student project at Chalmers University of Technology. This project is now over, but several of the original project members are still actively developing and maintaining HaskellDB. We do welcome new developers and patches, as all of us are full-time students.

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 (→ 4.2.11)

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

Further reading

http://haskelldb.sourceforge.net

4.4.6 ByteStream

Report by:	Bulat Ziganshin
Status:	beta, actively developed

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

4.4.7 Compression-2005

Report by:	Bulat Ziganshin
Status:	stable, actively developed

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, actively developed

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 (→ 4.4.5) 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!

Current work is directed at improving documentation and stability across platforms, and we hope to release the 1.0 version in October 2005, hopefully with SOEgraphics support.

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 built on top of wxHaskell (→ 4.5.1) which enables a concise and declarative programming style for *forms*: dialogs which only show and edit a set of values (used in many applications as *Options* or *Settings* dialogs). Control and layout definition are combined into one expression, there's no IO monad pro-

gramming, and values are passed to and from the controls almost automatically. Still, the type of the edited values and the layout structure can be managed independently, thanks to a programming technique called *compositional functional references*. As a new addition, a disjoint union type can be edited using a structure of radio buttons. Currently, FunctionalForms is in a proof-of-concept status, and not very actively developed further.

Further reading

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

4.5.3 Gtk2Hs – A GUI library for Haskell based on Gtk+

Report by:	Duncan Coutts
Maintainer:	Axel Simon and Duncan Coutts

This project provides a high-quality binding to Gtk+ which is a multi-platform toolkit for creating graphical user interfaces. GUIs written using Gtk2Hs follow the native look on Windows (since Gtk+ 2.6) and of course on Linux, Solaris and FreeBSD. Gtk+ and Gtk2Hs also supports MacOS X though it currently uses the X server and does not follow the native MacOS X theme.

Gtk2Hs also provides bindings to some Gnome extensions (at the moment Glade, GConf, a source code editor widget and a widget that embeds the Mozilla rendering engine). It also has automatic memory management (wxWidgets does not provide proper support for garbage-collected languages).

The Gtk2Hs library is actively maintained and developed. We are working towards a 1.0 release and are planning to support the new Cairo 2D graphics API.

We expect to release version 0.9.8 within the next few weeks. It will have GHC 6.4 compatibility, more extensive haddock (→ 5.5.6) reference documentation, 90% coverage of the Gtk+ API up to the latest version 2.6 and various minor bug fixes and API cleanups. It will also support a properties API in the same style as that of wxHaskell (→ 4.5.1)/Yampa.

Other changes since the last HCAR include a new website including many new screenshots. There is an introductory article by Kenneth Hoste in the first issue of The Monad.Reader (→ 1.5).

The current release of Gtk2Hs, version 0.9.7, is known to run on Linux, FreeBSD, MacOS X and Solaris. There was also a special 0.9.7.1 release specifically for Windows.

Packages are currently available for Windows, Fedora Core (→ 7.4.2), Gentoo (→ 7.4.4), Debian (→ 7.4.1), FreeBSD and ArchLinux.

Further reading

- http://haskell.org/gtk2hs/
- http://www.haskell.org/hawiki/TheMonadReader_2fIssueOne

4.5.4 HToolkit

Report by:	Krasimir Angelov

The HToolkit is a platform independent package for Graphical User Interfaces. The package is split into two libraries GIO and Port. The Port is a low-level Haskell 98+FFI (→ 3.2) compatible API, while GIO is a highlevel user friendly interface to Port. The primary goal of HToolkit is to provide a native look and feel for each target platform.

The currently supported platforms are Windows and Linux/GNOME.

There are some new things. There is a better support for menus and toolbars under both Windows and Linux. There is also new API which allows to create action based menu items and toolbar buttons. The "action" here is something like GtkAction widget but it is at Haskell level and it is available for both Windows and Linux. There isn't a new release yet.

Further reading

http://htoolkit.sourceforge.net/

4.5.5 HTk

Report by:	Christoph Lüth and George Russell
Status:	stable, actively maintained

HTk is an encapsulation of the graphical user interface toolkit and library Tcl/Tk for the functional programming language Haskell. It allows the creation of high-quality graphical user interfaces within Haskell in a typed, abstract, portable and concurrent manner. HTk is known to run under Linux, Solaris, FreeBSD, Windows (98, 2k, XP) and will probably run under many other POSIX systems as well. It works with GHC, version 6.0 and up to 6.4.

HTk is stable and actively maintained, but will not be developed further.

Further reading

http://www.informatik.uni-bremen.de/htk

4.5.6 Fudgets

Report by:	Thomas Hallgren

Fudgets is a GUI toolkit designed and implemented by Magnus Carlsson and Thomas. Most of the work was done in 1991–1995, and the library has been in minimal maintenance mode since then. It compiles with recent versions of GHC (e.g., GHC 6.2.1) on many Unix-like platforms (Linux, SunOS, Mac OS X, etc).

For documentation and downloads, see: http://www.cs.chalmers.se/Fudgets/.

Recent snapshots can also be found at: http://www.cse.ogi.edu/~hallgren/untested/.

Two applications using the Fudgets:

- The proof assistant Alfa, http://www.cs.chalmers.se/~hallgren/Alfa/
- The Programatica Haskell Browser, http://www.cse.ogi.edu/~hallgren/Programatica/

4.6 Graphics

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 (→ 3.2)
- 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 FunWorlds – Functional Programming and Virtual Worlds

Report by:	Claus Reinke
Status:	minimal update

FunWorlds is a currently mostly dormant experiment to investigate language design issues at the borderlines between concurrent systems, animated reactive 2&3d graphics, and functional programming. It built on the start that functional reactive programming and especially Conal Elliott's Fran made in that direction, but aimed for a simpler design and operational semantics, and thus more predictable performance.

In the yearly update to make the old snapshot build with the latest ghc (→ 2.1) (6.4 and the included HOpenGL/GLUT packages (→ 4.6.1)), lines and simple surfaces have been added to the scene graph primitives, together with a few simple examples demonstrating their use (animated parameterised surfaces, animated turtle graphics). This was in response to several recent enquiries about updates and further examples.

Further reading

- Project home: http://www.cs.kent.ac.uk/~cr3/funworlds/

4.7 Web and XML programming

4.7.1 HaXml

Report by:	Malcolm Wallace
Status:	stable, maintained

HaXml provides many facilities for using XML from Haskell. The public release is currently at version 1.12, soon to be refreshed to 1.13, mainly for compatibility with ghc-6.4, and to introduce support for building via Cabal (→ 4.1.1). Graham Klyne (→ 7.5.2) has a separate branch of 1.12, supporting namespaces, Unicode, and much more. We still hope eventually to merge those contributions back into the main HaXml tree.

Further reading

- http://haskell.org/HaXml
- http://www.ninebynine.org/Software/HaskellUtils/

4.7.2 Haskell XML Toolbox

Report by:	Uwe Schmidt
Status:	active development (current release: 5.01)

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),

The Haskell XML Toolbox bases on the ideas of HaXml (→ 4.7.1) and HXML, but introduces a more general approach for processing XML with Haskell. The Haskell XML Toolbox uses a generic data model for representing XML documents, including the DTD subset and the document subset, in Haskell. This data model makes it possible to use filter functions as a uniform design of XML processing applications. The whole XML parser including the validator parts was implemented using this design. Libraries with filters and combinators are provided for processing the generic data model.

Features

- validating XML parser
- very liberal HTML parser
- XPath support
- full Unicode support
- support for XML namespaces
- uniform data model for DTDs and XML content
- flexible arrow interface with type classes for XML filter
- hierarchical library support
- package support for ghc
- compatible with ghc-6.4
- 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

Current Work

- a project for supporting the Relax NG XML schema definition for validation ă is currently running and will be finished in September 2005.
- currently a master student works on a project developing a "HXT cookbook" for learning the application of the toolbox by example. In this user guide the development of a nontrivial example application in the context of RDF will be described. The programming technics with filters and their combinations and the arrow interface will be described on a real life problem.

Further reading

The Haskell XML Toolbox Webpage (http://www.fh-wedel.de/~si/HXmlToolbox/index.html) includes downloads, online documentation and a master thesis describing the design of the toolbox. The documentation is a bit out of date. This is one reason for the users guide project.

4.7.3 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:

- package-ifycation of WASH (& much simpler installation)
- caching of documents (but turned off by default)

Current work includes

- 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 (→ 4.2.11)) as well as the implementation of a session as a continually running server thread
- 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.

4.7.4 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 [1]. 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 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 [2].

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.

[1] Scrap your boilerplate with class: extensible generic functions, Ralf Lämmel and Simon Peyton Jones (2005) - submitted to ICFP 2005.

[2] Implementing Web-Services with the HAIFA Framework. Simon Foster (2005). In Monad.Reader issue 1.

4.7.5 Haskell XML-RPC

Report by:	Björn Bringert
Status:	maintained

Haskell XML-RPC 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.

Further reading

http://www.bringert.net/haskell-xml-rpc/

5 Tools

5.1 Foreign Function Interfacing

5.1.1 C–>Haskell

Report by:	Manuel Chakravarty
Status:	active

C–>Haskell is an interface generator that simplifies the development of Haskell bindings to C libraries. Development in the past year has concentrated on stabilising the current feature set. Source and binary packages as well as a reference manual are available from http://www.cse.unsw.edu.au/~chak/haskell/c2hs/.

5.1.2 JVM Bridge

Report by:	Ashley Yakeley
Status:	stalled

JVM-Bridge is a GHC package intended to allow full access to the Java Virtual Machine from Haskell, as a simple way of providing a wide range of imperative functionality. Its big advantage over earlier attempts at this is that it includes a straightforward way of creating Java classes at run-time that have Haskell methods (using DefineClass and the Java Class File Format). It also features reconciliation of thread models without requiring GPH.

Current Status

JVM-Bridge is at version 0.3: it works on Windows and also allows the use of third-party Java libraries. A 0.3.1 release to fix Mac OS X build issues may be forthcoming.

Further reading

http://sourceforge.net/projects/jvm-bridge/

5.2 Scanning, Parsing, Analysis

5.2.1 Alex version 2

Report by:	Simon Marlow
Status:	stable, maintained

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

Status: No change since the last report. The latest version is 2.0, released on August 13, 2003. Alex is in maintenance mode at the moment, and a few minor bugs reported since 2.0 have been fixed in CVS. A minor release will probably be made at some point.

Further reading

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

5.2.2 Happy

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

Paul's Generalized LR (GLR) extension for Happy has now been 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. Paul has converted the GHC Haskell grammar to a GLR parser, but is still experimenting with the result. Recently, the Chalmers BNFC tool (→ 5.5.7) has been updated to work with the GLR mode of Happy.

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/.

5.2.3 HaLex

Report by:	Jorge Sousa Pinto

HaLeX is a Haskell library to model, manipulate and animate regular languages. This library introduces a number of Haskell datatypes and the respective functions that manipulate them, providing a clear, efficient and concise way to define, to understand and to manipulate regular languages in Haskell. For example, it allows the graphical representation of finite automata and its animation, and the definition of reactive finite automata. This library is described in the paper presented at FDPE'02.

5.2.4 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.

5.2.5 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)).

5.2.6 SdfMetz

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

SdfMetz supports grammar engineering by calculating grammar metrics and other analyses. It reads SDF grammar specification files and 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. 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 (→ 7.3.8).

5.2.7 HaGLR

Report by:	Jorge Sousa Pinto and Joost Visser

HaGLR is an implementation of Generalized LR parsing in Haskell. Apart from parsing with the GLR algorithm, it supports parsing with the LR algorithm, visualization of deterministic and non-deterministic finite automata, and export of ASTs in XML or ATerm format. As input, HaGLR accepts either plain context-free grammars, or SDF syntax definitions. The SDF front-end is implemented as an extension of the Sdf2Haskell generator (→ 5.2.5). HaGLR's functionality can also be accessed as library functions, available under the Language.ContextFree subdivision of the UMinho Haskell Libraries (→ 7.3.8). HaGLR was implemented by João Fernandes and João Saraiva.

Further reading

HaGLR is available from http://wiki.di.uminho.pt/twiki/bin/view/PURe/HaGLR.

5.2.8 DrHylo

Report by:	Jorge Sousa Pinto

DrHylo is a tool for deriving hylomorphisms from Haskell program code. Currently, DrHylo accepts a somewhat restricted Haskell syntax. It is based on the algorithm first presented in the paper Deriving Structural Hylomorphisms From Recursive Definitions at ICFP'96 by Hu, Iwasaki, and Takeichi. To run the programs produced by DrHylo, you need the Pointless library.

Further reading

DrHylo is available from http://wiki.di.uminho.pt/bin/view/Alcino/DrHylo.

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 (→ 4.3.1), 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

- 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 release supports the hierarchical modules extension, but only small parts of the hierarchical libraries, unfortunately.

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 (→ 5.5.6) documentation. Please let us know if you are using the transformations.

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. We very much welcome user feedback, currently especially on the new API and our recent "styles of monadification" survey, available from http://www.cs.kent.ac.uk/projects/refactor-fp/Monadification.html.

A snapshot of HaRe is available from our web page, as are recent presentations from the group (including LDTA 05), and an overview of recent work from staff, students and interns.

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).

5.3.5 LVM-OPT

Report by:	Eelco Visser and Jory van Zessen

Optimization of functional programs through strategic program transformation is still one of the projects we are pursuing in the Stratego/XT group, even if it is at a slow pace. After a year of silence, Jory van Zessen has taken over the stick from Alan van Dam and the HsOpt (see November 2003 edition of the HC&A Report) project has become the lvm-opt project; the target of optimization is LVM code produced by the Helium compiler. The goal remains to create a (full-blown) simplifier/optimizer for a lazy functional language based on rewrite rules controlled by strategies.

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, for graphical display and animation, enable the user to observe the values of selected expressions in a program. Hood and GHood are easy to use, because they are based on a small portable library. A variant of Hood is built in to 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

CHD: http://www.informatik.uni-kiel.de/~fhu/chd/

5.4.2 Hat

Report by:	Olaf Chitil and Malcolm Wallace
Status:	several recent additions; stable release forthcoming

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 with several tools in various ways: Hat-observe provides observation of functions. Hat-trail enables backwards exploration of a computation, starting from (part of) a faulty output or an error message. Hat-detect is an algorithmic debugger very similar to Buddha (→ 5.4.3). Hat-explore provides a user interface similar to traditional source-based debuggers, but allows free navigation through the trace and incorporates algorithmic debugging and program slicing. Hat-cover highlights the parts of the source program that have been executed during the computation. Hat-anim is a forward-animator showing the reduction sequence of expressions. We also have prototypes of two tools for extracting diagnostic paths from non-terminating computations. If the computation dives into a black hole, black-hat can be used; for other forms of non-productive non-termination hat-nonterm can be used. All tools inter-operate and use a similar command syntax.

A tutorial explains how to generate traces, how to explore them, and how they help to debug Haskell programs. Hat can be used both with nhc98 and ghc, and can also be used for Haskell 98 programs that use some language extensions (FFI (→ 3.2), MPTC, fundeps, hierarchical libs).

We expect to release a new public version 2.04 of Hat within a few days. This will contain numerous bugfixes, several new features and the new prototype viewing tools mentioned above. In particular, it will work more smoothly with ghc-6.4.

Further reading

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.

Since the last report buddha has not had an official release. The next release will be delayed further while I write my thesis.

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

A paper about buddha appears in the proceedings of the Advanced Functional Programming Summer School, which was held in Tartu, August 2004.

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, hbc, or nhc98 – except jhc (which does not compile modules separately anyway). The public release has recently been refreshed to version 3.10. Occasional maintenance and bugfixes continue to the CVS tree at haskell.org.

Further reading

http://haskell.org/hmake

5.5.2 cpphs

Report by:	Malcolm Wallace
Status:	active development

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 0.9, and is pretty stable.

Further reading

http://haskell.org/cpphs

5.5.3 Visual Studio support for Haskell

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

The Visual Studio project is making great progress, thanks to Krasimir Angelov who is working on Visual Studio during his internship at Microsoft Research Cambridge.

We plan to make a first release of the Visual Studio extension in about two months' time. Our current testing version has many improvements over earlier versions:

- The editor the usual features: syntax colouring and detection of errors on the fly, and jumping to the declaration for a function or type.

  It also includes pop-up information on all identifiers in the source module: the type of a function name, or definition of a type constructor or class, for example. This even works in a multi-module program, and includes all of the GHC extensions, thanks to our use of the GHC API underneath.

  Krasimir has also implemented a program browser, which lists all the functions, types and classes for each module in the project, from which you can quickly jump to the source code for any entity. Again, because we're using the GHC API underneath, the information is updated interactively as you edit the program without requiring a separate compilation step.

- Support for multi-module programs and libraries ("projects" in VS terminology) is now more mature, and is based entirely on Cabal (→ 4.1.1). A Cabal package can be loaded into VS, and VS projects are saved as fully-fledged Cabal projects, which can be built on a machine that doesn't have Visual Studio installed. Cabal is used as the underlying build system in VS.

- The installer bundle contains Visual Studio plugin together with GHC, Alex (→ 5.2.1), Happy (→ 5.2.2) and Haddock (→ 5.5.6) so with Visual Haskell you will have complete development environment for Haskell.

- The GHC libraries documentation and user manual are integrated in the combined Visual Studio help collection.

Help is 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⟩).

5.5.4 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. This 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 0.7 (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 (→ 5.5.6), and launching from the IDE. In the time between the last HC&A report and now some experimentation with the more language-aware features of Eclipse IDEs took place. There is now an experimental refactoring support (Rename Module) and a basic implementation of a Content Outliner.

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

5.5.5 haste

Report by:	Rickard Nilsson
Status:	active development

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

5.5.6 Haddock

Report by:	Simon Marlow
Status:	stable, maintained

The latest release is version 0.6, released November 11 2003.

Since then, various updates have been incorporated into the source tree, and I plan to release version 0.7 before too long.

The main improvements are:

- Overhaul of the way Haddock decides where to point hyperlinks for imported entities. Previously, the hyperlink would point to the documentation for the entity in the module it was imported from. This turned out not to work well in some cases, so we switched to a more global approach: Haddock now determines a single "home location" for each entity, and every hyperlink for that entity will always point to the home location. The home location is currently defined as the "lowest" module in the module dependency graph, that is not hidden. This definition seems to work reasonably well: check out the hyperlinks in the documentation for the GHC 6.4 libraries, for example.
- Sections of documentation can be collapsed and expanded with the usual +/− buttons. This is particularly useful for instances, which often take up a lot

of space. The module hierarchy on the contents page now has +/− buttons for collapsing subtrees.

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/

5.5.7 BNF Converter

Report by:	Markus Forsberg
Status:	active

BNF Converter is a multi-lingual compiler tool. BNFC takes as its input a grammar written in LBNF (Labelled BNF) notation, 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++ and their standard parser and lexer tools. It also supports XML generation.
BNFC itself was written in Haskell.
Source code and Documentation can be downloaded at the BNFC homepage.
BNF Converter is a package in Debian Linux (→ 7.4.1).

Further reading

http://www.cs.chalmers.se/~markus/BNFC

5.5.8 Hoogle – Haskell API Search

Report by:	Neil Mitchell
Status:	beta, in progress

Hoogle is a 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. When a function is found, the API documentation given at Zvon (http://www.zvon.org/other/haskell/Outputglobal/) is displayed.
Hoogle is still very much in its early stages. While it supports some types of type isomorphisms, it often gets it wrong. It has basic support for finding functions with more general types, and those with missing or reordered arguments. Enhancements planned include removal of bugs, support for type classes, a downloadable console based version and the addition of more libraries. Of course, this tool is written in Haskell.

Further reading

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

6 Applications

6.1 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 interpreter, and compiler backends targetting both GHC and the Parrot virtual machine. It also supports inline Haskell code in Perl 6 modules, as well as dynamic Haskell evaluation through the hs-plugins (→ 4.2.11) package.
As of this writing, we are on the 6.2.x release series, working toward Perl 6's object-oriented model. Recently we have also switched to base on GHC 6.4, taking full advantage of GADTs, STM, and improved Template Haskell features.
The Pugs team has over 60 committers from both Haskell and Perl worlds. 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/

6.2 HScheme

Report by:	Ashley Yakeley
Status:	stalled

HScheme is a project to create a Scheme interpreter written in Haskell. There's a stand-alone interpreter program, or you can attach the library to your program to provide "Scheme services". It's very flexible and general with types, and you can pick the "monad" and "location" types to provide such things as a purely functional Scheme, or a continuation-passing Scheme (that allows call-with-current-continuation) etc.

Current status

There's an online interpreter that I keep up to date. There are a couple of major issues that stand before R5RS compliance, after which I'll make a release. See http://hscheme.sourceforge.net/issues.php.

Very little work is currently being done on it.

Further reading

http://hscheme.sourceforge.net/

6.3 Darcs

Report by:	David Roundy
Status:	active development

Darcs is a distributed revision control system (i.e., CVS replacement), written in Haskell. In darcs, every copy of your source code is a full repository, which allows for full operation in a disconnected environment, and also allows anyone with read access to a darcs repository to easily create their own branch and modify it with the full power of darcs' revision control. Darcs is based on an underlying theory of patches, which allows for safe reordering and merging of patches even in complex scenarios. For all its power, darcs remains very easy to use tool for every day use because it follows the principle of keeping simple things simple.

Darcs version 1.0.3rc1 was released on April 25, 2005. The latest release has a number of interface enhancements contributed by several developers, along with the usual bugfixes. We've recently had a bit of a reorganization of responsibilities, with Tomasz Zielonka taking over maintainership of the stable branch of darcs, with Ian Lynagh taking over maintainership of the unstable branch.

Recently we have been working to make darcs scale efficiently to large repositories, which is an interesting challenge, making sure we have just the right amount of laziness. There's also ongoing work to make darcs interoperate with git, and as always there are interface improvements in the works. David Roundy has been focusing on a new simple and more efficient framework for handling conflicts.

As always, it's a good time to join darcs development if you're looking for a place to apply your haskell expertise. We've got a whole list of wishlist features waiting for developers!

Darcs is free software licensed under the GNU GPL.

Further reading

http://darcs.net

6.4 FreeArc

Report by:	Bulat Ziganshin
Status:	beta, actively developed

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 (→ 4.4.7), ByteStream (→ 4.4.6), Process (→ 4.2.1)). 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

6.5 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 http://www.cin.ufpe.br/~haskell/hwsproxygen. The project was created by the Informatics Centre of Federal University of Pernambuco. Extensions and enhancements are welcome.
The immediate plans are to write an English paper for HWSProxyGen, in order to deeply reach the Haskell community. 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.) (→ 4.7.5)
  http://www.bringert.net/haskell-xml-rpc/http.html

6.6 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 (→ 6.7). The last release is still version 0.3, though I have various bug fixes and improvements that I should release soon, including basic text search and improved channel nicks handling. I would like to find time to work on adding auto-reconnection and support for multiple-servers to make it more useful. There should probably also be a menubar. Contributions are most welcome.

Further reading

http://haskell.org/hircules/

6.7 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. You can download lambdabot from the darcs repo here:
The source repository is available:

```
darcs get
    http://www.cse.unsw.edu.au/~dons/lambdabot
```

6.8 Flippi

Report by:	Philippa Cowderoy

Flippi is a lightweight (and currently somewhat underfeatured) wiki clone written in Haskell and released under the BSD license. The current release is v0.03, which added support for scripting and a RecentChanges script. The main planned feature for the next release is a template facility, with various interface alterations in the default setup being likely. Also in the pipeline is a refactoring of the parser to make adding new pieces of markup syntax easier, and metadata support and revision histories with reversion are planned by v0.1.

A goal in development so far, and one which the author would like to maintain, is to keep the code easy to understand and modify – to this end, the configuration is currently all done by source modification. This isn't necessarily as bad as it sounds – if the Flippi CGI is run via runhugs or similar, there's no perceivable difference to somebody configuring Flippi bar the level of power available. However, so far Flippi has only been tested under GHC 6.2 and is dependant on a recent version of the hierarchical libraries.

Further reading

- http://www.flippac.org/projects/flippi/
- http://www.scannedinavian.org/cgi-bin/flippi/flippi

6.9 Postmaster ESMTP Server

Report by:	Peter Simons

Postmaster is an Internet mail transport agent (MTA) written and configured in Haskell. At the time of this writing, it handles incoming ESMTP network connections and delivers accepted messages to the user's mailbox by piping it into an arbitrary local mailer (e.g. Procmail).

As is to be expected from an MTA written in Haskell, it is configurable beyond anything you'll ever need. The server itself comes as a monadic combinator library; so you can plug together or modify the components as you please. A pretty sophisticated standard configuration on which to build is part of the distribution.

Postmaster is still very young; there remains a lot to be done before it can really compete with Sendmail or Postfix. Most notably, it lacks any form of queue management right now. Nonetheless, for leaf sites, which don't need to do extensive mail relaying, it is a reliable and powerful solution already.

Further details are available at: http://postmaster.cryp.to/

It is worth noting that Postmaster includes several generally useful libraries which are not tied to the ESMTP server:

BlockIO implements a monad for fast, non-blocking I/O with static `Ptr Word8` buffers.

HsDNS implements an asynchronous DNS resolver an top of the GNU adns library.

HsEMail Parsec (→ 4.3.1) parsers for most of RFC 2821 and 2822.

Child provides `spawn`, `par`, and `timeout` for more flexible handling of child computations started with `forkIO`.

Syslog FFI (→ 3.2) bindings to the `syslog(3)` system API.

hOpenSSL (very incomplete) FFI bindings to the OpenSSL library. At the moment provides mostly access to the `libcrypto` part.

6.10 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.

The latest version of Riot was released on 2005-05-06 and includes support for configuration files through hs-plugins (→ 4.2.11) and other minor improvements.

Further reading

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

6.11 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 configurable syntax highlighting, and emacs emulation.

The source repository is available:

```
darcs gethttp://www.cse.unsw.edu.au/~dons/yi
```

6.12 Dazzle (formerly NetEdit)

Report by:	Martijn Schrage and Arjan van IJzendoorn

Dazzle is a graphical editor 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.

Further reading

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

6.13 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.

6.14 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.
- Hopefully, somewhere around April 2005 will appear the first public release of the program Dumatel-1.02, a prover based on term rewriting and equational reasoning (written in Haskell).

Further reading

http://haskell.org/docon/

6.15 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: it handles my complete Ph.D. thesis without any problems, and I see that Graham Hutton makes use of lhs2TEX in his new book (→ 1.6.1).

There has not been a release for quite some time, but I still hope to make one soon. Development continues in the Subversion repository.

Further reading

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

6.16 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/

6.17 Converting knowledge-bases with Haskell

Report by:	Sven Moritz Hallberg

In November, 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 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.

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 and file store 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 Spring 2005, Galois is 17 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're stepping up our community involvement: cvs.haskell.org is about to move to a new, much beefier machine that will be funded and maintained by Galois. We'll also be supporting various community efforts on that machine, such as the Hackage database and The Haskell Sequence.

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: Mark Carroll

Aetion Technologies LLC is a small business located in central Ohio, USA, developing commercial uses of generic artificial intelligence software. Much our our codebase and development is done in Haskell using GHC. We often test the feasibility of new applications, and Haskell seems excellent for rapidly producing prototypes that work well. Principal areas of business are model-based inference and decision making for the military and for finance. Code we write that is not part of our core software, but instead rather more generic, we will tend to release under an open source license. Over this year we are recruiting more programmers, and we tend to require applicants to send Haskell code samples before interviewing them seriously.

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 teached 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 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 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, to obtain more inference rules for equality in call-by-need lambda-calculi, e.g. a definition of behavioural equivalence. *Matthias Mann* has established a proof of its soundness w.r.t. contextual equivalence for a non-deterministic call-by-need lambda calculus. Further research is aimed towards extensions of this calculus to support work on strictness analysis using abstract reduction.

Strictness Analysis using Abstract Reduction

The algorithm 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 2004 we proved correctness of the algorithm by using a non-deterministic call-by-need lambda-calculus. A technical report covering the latter is available from our website. The proof of correctness of strictness analysis using abstract reduction uses a conjecture that the defined behavioural equivalence is included in the contextual equivalence.

A current result is a reformulation of the proof that uses a deterministic call-by-need lambda calculus and that does not depend on the above mentioned conjecture.

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

7.3.2 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).

The MMiSS project has developed a repository providing configuration management, version control and change management for semantically structured documents. It holds teaching material for over 20 courses in the domain of safe and secure system development. The implementation comprises over 100k lines of Haskell code.

We are further 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, CSP-CASL 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 (→ 3.5), Haddock (→ 5.5.6), the combinator library Parsec (→ 4.3.1), HaXml (→ 4.7.1) and Programatica (→ 5.3.1).

Further reading

- Group activities overview:
  http://www.informatik.uni-bremen.de/agbkb/forschung/formal_methods/
- MMiSS Multimedia instruction in safe systems:
  http://www.mmiss.de
- CASL specification language:
  http://www.informatik.uni-bremen.de/cofi
- Heterogeneous tool set:
  http://www.informatik.uni-bremen.de/cofi/hets

7.3.3 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 continue our work on debugging GUM in our version of Parallel Haskell. We have recently completed

reimplementation of a serial version of a quantum simulator, and we are going to parallelize it for research on quantum algorithms.

Further reading

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

Contact

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

7.3.4 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:

- A literate version of the nhc98 runtime is being produced to form the basis of porting efforts (→ 3.1.1) and DSL experiments.
- Qingsong Ye has finished his Masters project that developed an embedded DSL for mobile device synchronisation. A paper on this work will appear later this year at the Fourth International Conference on Mobile Business.
- Yun-Hsian Wu has submitted a project that investigated the use of embedded DSLs to produce GUIs for handheld devices.
- Matt Roberts is looking at the use of commercial GPU hardware to run functional programs.

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/.

7.3.5 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 (→ 4.3.4). Huiqing Li, Simon Thompson and Claus Reinke have released further snapshots of HaRe, the Haskell Refactorer (→ 5.3.3). Yong Luo recently joined the group to work with Olaf on theoretical foundations of tracing and together with Thomas Davie and the York functional programming group the Haskell tracer Hat is extended and improved further (→ 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://gtk2hs.sourceforge.net/
- 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/

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

Report by:	Phil Trinder
Members:	Abyd Al Zain, Andre Rauber Du Bois, 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 OGSA.

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 (→ 3.3.1) and Parallel ML with skeletons (PMLS). Distributed/mobile languages include Glasgow distributed Haskell (→ 3.3.2), Erlang (http://www.erlang.org/), Hume (http://www-fp.dcs.st-and.ac.uk/hume/), JoCaml and Camelot.

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/

7.3.7 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 (→ 5.1.1) interface generator and more recently the hs-plugins (→ 4.2.11) library for dynamically loaded type-safe plugins. As a testbed for further research in dynamic code loading and type checking, we are developing a highly customisable editor in Haskell, called Yi (→ 6.11). We also recently released PanTHeon, a portable re-implementation of Conal Elliott's Pan animation tool based on meta-programming in Template Haskell.

In cooperation with Microsoft Research, Cambridge, we recently proposed *associated types* for Haskell type classes. Associated types are a form of type-indexed data types realised as data declarations in classes, which facilitate some forms of generic programming. We are currently in the process of implementing this extension in the Glasgow Haskell Compiler.

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

7.3.8 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 point-free programming in Haskell (→ 4.2.10), as well as the DrHylo tool (→ 5.2.8) 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 (→ 5.2.4), 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 (→ 5.2.5), 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.

7.3.9 The Computer Systems Design Laboratory at the University of Kansas

Report by: Perry Alexander

The Computer Systems Design Laboratory at the University of Kansas is using Haskell in several distinct projects.

We are continuing work, previously reported in the Communities and Activities report, developing tools for the Rosetta specification language. This work uses Haskell as the primary platform for a toolset facilitating the analysis of heterogeneous models written in Rosetta. We use a composable interpreter framework to provide basic language interpretation and a collection of static and dynamic analysis tools.

A related project utilizes Haskell in the development of a generalized proof assistant, Prufrock. This provides a framework for integrating new languages into the proof environment. The language representation is separated from the logical inference rules, using generic programming techniques. Proof tactics (written in Haskell), interaction, and specific prover implementation, including such features as global state and logging are separated using Haskell's type class system. This results in a set of (largely) independent modules that can be combined to produce a specialized first-order theorem prover for a given language and a given system of inference. A technical report, including the entire Prufrock source, is available at the Prufrock website.

Another project explores the implementation of functional languages, via graph reduction, on FPGA hardware. This system combines a compiler and simulator, written exclusively in Haskell, with a VHDL implementation of the abstract machine. A graph reduction machine is being synthesized in FPGA from VHDL source to directly execute compiled Haskell.

Finally, we are Haskell to development a formalism to represent dance. Currently, choreographers communicate only by performance. The dance language aims to provide a mechanism for to allow choreographers a means to communicate the structure of a routine textually. Additionally, the dance language has an associated typechecker, used by choreographers to detect errors in the transcription of routines.

Further reading

- Computer Systems Design Lab: http://www.ittc.ku.edu/research/view_lab.phtml?lab=CSDL
- Systems Level Design Group: http://www.ittc.ku.edu/Projects/SLDG/
- Rosetta Specification Language: http://www.sldl.org
- Prufrock: http://www.ittc.ku.edu/~wardj/prufrock/
- Dance Language: http://www.ittc.ku.edu/~jenis/

7.3.10 Cover: Combining Verification Methods

Report by: Patrik Jansson
Participants: John Hughes, Thierry Coquand, Peter Dybjer, Mary Sheeran, Marcin Benke, Koen Claessen, Patrik Jansson, Andreas Abel, Gregoíre Hamon, Ulf Norell, Fredrik Lindström, Nils Anders Danielsson

Cover is a Haskell-centered research project at Chalmers funded by the Swedish Foundation for Strategic Research. The goal is to develop methods for improving software quality. The approach is to integrate a variety of verification methods into a framework which permits a smooth progression from hacking code to fully formal proofs of correctness.

More concretely we work on these components:

- QuickCheck – automated random testing (→ 5.4.4)
- Agda – a dependently typed language and its proof engine (implemented in Haskell) (→ 3.4.1)
- Cover Translator – translation from Haskell to
  - Agda – for interactive proof
  - First order logic – for automated proof

Our best results so far include:

- Development of QuickCheck (automatic shrinking, monadic testing, etc.)
- Development of Agda (built-in types, a class system, "implicit arguments", etc). Development of methodology for reasoning about general recursive programs.
- Cover Translator. Translates Haskell (via GHC Core) into a first order formulas understood by automatic theorem provers such as Gandalf and Vampire. Ongoing work on case studies.
- A first prototype - an extension of Alfa (an advanced GUI for the prover Agda) with tools for testing and automatic proof construction.
- Agsy – automatic proof search plugin for Agda.
- Collaboration with AIST (Advanced Industrial Science and Technology Institute in Japan) on development and applications of Agda.

The Cover source code can be browsed at http://cvs.coverproject.org/marcin/cgi/viewcvs/ and can be accessed by anonymous CVS from cvs.coverproject.org.

Short term goals:

- Complete the chain of tools (QuickCheck, Agda, Cover Translator) so that we can take actual Haskell code, test, translate for the theorem provers, and prove properties.
- Identify larger scale case studies that ought to be tractable for our methods.

Further reading

For more details about the project, read about QuickCheck (→ 5.4.4) and Agda (→ 3.4.1) in this report or consult the homepage at http://coverproject.org.

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 package repositories of selected Haskell projects for Fedora Core. At the time of writing there are packages for c2hs-0.13.4, darcs-1.0.2, DrIFT-2.1.1, ghc-6.4, greencard-3.01, gtk2hs-0.9.7, haddock-0.6, happy-1.15, hircules-0.3, hs-plugins-0.9.8, hugs98-Mar2005, wxhaskell-0.9 for i386 and x86_64. There is a mailing list ⟨fedora-haskell@haskell.org⟩ for announcements and questions. Contributions are most welcome. I am still hoping to get ghc added to Fedora Extras soon.

Further reading

http://haskell.org/fedora/

7.4.3 OpenBSD Haskell

Report by: Don Stewart

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

GHC is available for i386 and amd64. nhc98 is available for i386, sparc and powerpc. Hugs 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, happy, haddock and darcs.

7.4.4 Haskell in Gentoo Linux

Report by: Andres Löh

A lot has been happening behind the scenes in the Gentoo/Haskell world. Two new developers, Luis Araujo and Duncan Coutts, have recently joined the Haskell team.

We just have unmasked ghc-6.4 a few days ago. Some libraries are still lacking support, but we hope that this will change in the next few months. The eclass that supports the registration of ghc packages has been improved and updated for ghc-6.4; it seems to be working fine.

We internally use a darcs (→ 6.3) overlay to exchange and test new ebuilds, where we are currently preparing an eclass to make ebuild-writing for Cabalized packages (→ 4.1.1) a triviality.

We have the following long-term goals:

- improve support for non-x86 platforms,
- improve support for nhc98 and Hugs (and potentially jhc (→ 2.4) in the future),
- add more libraries and tools.

New ebuilds, comments and suggestions, and bug reports can be filed at bugs.gentoo.org. Make sure that you mention "Haskell" in the subject of the report. Or visit us on IRC (`#gentoo-haskell` on freenode).

7.5 Individuals

7.5.1 Oleg's Mini tutorials and assorted small projects

Report by: Oleg Kiselyov

The page about type system hacks (http://pobox.com/~oleg/ftp/Haskell/types.html) – a part of the collection of various Haskell mini-tutorials and assorted small projects (http://pobox.com/~oleg/ftp/Haskell/) – has received two additions:

Genuine keyword arguments

We show the Haskell implementation of keyword arguments, which goes well beyond records (e.g., in permitting the re-use of labels). Keyword arguments indeed look just like regular, positional arguments. However, keyword arguments may appear in any order. Furthermore, one may associate defaults with some keywords; the corresponding arguments may then be omitted. It is a *type error* to omit a required keyword argument. The latter property is in stark contrast with the conventional way of emulating keyword arguments via records. Also in marked contrast with records, keyword labels may be *reused* throughout the code with no restriction; the same label may be associated with arguments of different types in different functions. Labels of Haskell records may not be re-used. Our solution is essentially equivalent to keyword arguments of DSSSL Scheme or labels of OCaml.

How to write an instance for not-a-function

There are occasions when one wishes to write a class instance for un-function: For example, an instance of a class `class Vspace a v | v -> a` that applies *only* when the second type argument is not of a functional type. Overlapping instances per se do not help because a more general instance nominally applies whenever a more specialized instance does, which will contradict the functional dependency. Our solution is a class `IsFunction v f` where the second parameter is uniquely determined by the first. To be precise, the second parameter is inferred to be of the type `HTrue` if and *only if* the first parameter is an arrow type, `x -> y` for some `x` and `y`. The second parameter of `IsFunction` is inferred to be of the type `HFalse` in any other case. The solution can be easily generalized to implement *negation* of any other structural type constraint (tuple, list, `Maybe`, etc).

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 Alain Crémieux

Report by: Alain Crémieux

I am working on a port to Haskell of a compiler for the Tiger toy language. The reference for the Tiger language is the book from Andrew Appel "Modern compiler implementation in ML". The corresponding code in ML is available on the Web, written by Yu Liao.

So the first step is a port of this code, but with the use of Haskell's tools Alex and Happy, up to the generation of machine code for a RISC processor. Then there will be 2 new outputs for the compiler, one directed at C– (a Tiger compiler generating C– is available, written in O'CAML by Paul Govereau), and the other towards LLVM, a virtual machine system (some examples of a Tiger compiler link to LLVM, written by Chris Lattner, are also available).

The roadmap is to add an intelligent editor for Tiger, in the style of Helium, and some kind of source level debugger, by reusing available components. All this should lead to a practical example of the complete implementation of a language, which is a domain in which Haskell is especially good. Any help & suggestions welcome, of course.

7.5.4 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. Recent case-studies include

- mixtures of time-series,
- Bayesian networks,
- time-series models and "the" sequence-alignment algorithm.

(A spring-clean of the code is overdue.)

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? The best name suggested so far is 'inductive programming'.

I am currently developing the time-series models further, and must also look at Template-Haskell or something 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/

7.6 Bioinformatics tools

Report by:	Ketil Malde

As part of my Ph.D 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 in the process of redesigning/reimplementing the algorithms, hopefully leading to a more integrated, flexible and efficient toolset.

Further reading

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

7.6.1 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.

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.

Further reading

http://www.math.duke.edu/~wgm