Haskell Communities and Activities Report

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

Eleventh Edition – November 30, 2006

Andres Löh (ed.)

Lloyd Allison, Tiago Miguel Laureano Alves, Krasimir Angelov,
Dmitry Astapov, Alistair Bayley, Jean-Philippe Bernardy,
Clifford Beshers, Edwin Brady, Chris Brown,
Andrew Butterfield, Manuel Chakravarty, Olaf Chitil,
Alain Crémieux, Jácome Cunha, Iavor Diatchki,
Atze Dijkstra, Robert Dockins, Frederik Eaton,
Martin Erwig, Simon Foster, Leif Frenzel,
Richard A. Frost, Andy Gill, Dimitry Golubovsky,
Murray Gross, Walter Gussmann, Keith Hanna,
Ralf Hinze, Paul Hudak, Liyang Hu,
Graham Hutton, S. Alexander Jacobson, Johan Jeuring,
Antti-Juhani Kaijanaho, Jeremy O'Donoghue, Oleg Kiselyov,
Marnix Klooster, Lennart Kolmodin, Eric Kow,
Lemmih, Huiqing Li, Andres Löh,
Rita Loogen, Salvador Lucas, Ketil Malde,
Simon Marlow, Conor McBride, Serge Mechveliani,
Arie Middelkoop, Neil Mitchell, William Garret Mitchener,
Andy Adams-Moran, J. Garrett Morris, Yann Morvan,
Diego Navarro, Rishiyur Nikhil, Sven Panne,
Ross Paterson, Jens Petersen, Simon Peyton-Jones,
Bernie Pope, Claus Reinke, Colin Runciman,
Alberto Ruiz, David Sabel, Uwe Schmidt,
Martijn Schrage, Alexandra Silva, Axel Simon,
Anthony Sloane, Dominic Steinitz, Donald Bruce Stewart,
Glenn Strong, Martin Sulzmann, Doaitse Swierstra,
Wouter Swierstra, Audrey Tang, Henning Thielemann,
Peter Thiemann, Simon Thompson, Phil Trinder,
Arjan van IJzendoorn, Miguel Vilaca, Joost Visser,
Malcolm Wallace, Stefan Wehr, Ashley Yakeley,
Bulat Ziganshin

Preface

Welcome to the eleventh edition of the Haskell Communities and Activities Report – a collection of entries about everything that is going on and related to Haskell in some way that appears twice a year.

This time, there was a rather long delay between the submission deadline and the actual publication. I apologize for any entries that are already outdated now due to this delay and promise that I will try to keep this time shorter for the next edition.

I have looked at the total number of entries in all the reports and discovered that the report has been continuously growing until the 11/2005 edition, which had 149 entries. After that, there was a decline: the 06/2005 edition had 144 entries, and the current 11/2006 edition has only 134 entries.

How and why do entries get removed? Generally, entries have to be updated every time for inclusion in the report. If I do not hear back from the author of an entry or there are no updates, but the entry seems still up-to-date, I keep it around for the next edition. But every entry should be updated in a more or less significant way at least once per year, otherwise it is removed. The idea of this Report is to document the current state of the communities, after all, and of course an old entry can be revived if there is activity again.

I hope that the recent decline in entries is a temporary phenomenon and not the beginning of a long-term trend. All the more, I want to thank all the authors who have found the time to update their entries or provide new reports! As always, I have enjoyed putting everything together and reading about all the exciting new developments.

There are certainly some projects that are missing from this Report. When I am aware of a certain project or a recent development, I often try to encourage the authors to submit in time, but I also rely on you, the readers, to ask missing projects to provide information for the next edition, or to write about your own project. Do not worry about whether your project is adequate: If it has anything to do with Haskell at all, there is a place for it in the Report.

Please mark the final weeks of April in your calendar, because that is when the entries for the May edition will be collected.

As always, feedback is very welcome ⟨hcar@haskell.org⟩. Enjoy the Report!

Andres Löh, University of Bonn, Germany

Contents

1 General

1.1 HaskellWiki and haskell.org

Report by: Ashley Yakeley

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

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

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

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

Further reading

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

1.2 #haskell

Report by: Don Stewart

The `#haskell` IRC channel is a real-time text chat where anyone can join to discuss Haskell. The channel has grown substantially in users over the last 6 months, and now `#haskell` averages over 240 concurrent users. Point your IRC client to irc.freenode.net and join the `#haskell` conversation!

For non-English conversations about Haskell there is now:

- `#haskell.de` – German speakers
- `#haskell.es` – Spanish speakers
- `#haskell.fi` – Finnish speakers
- `#haskell.fr` – French speakers
- `#haskell.hr` – Croatian speakers
- `#haskell.it` – Italian speakers
- `#haskell.jp` – Japenese speakers
- `#haskell.se` – Swedish speakers
- `#haskell_ru` – Russian speakers

Related Haskell channels are now emerging, including:

- `#haskell-overflow` – Overflow conversations
- `#haskell-blah` – Haskell people talking about anything except Haskell itself
- `#gentoo-haskell` – Gentoo/Linux specific Haskell conversations (→ 7.4.3)
- `#darcs` – Darcs revision control channel (written in Haskell) (→ 6.4)

Further reading

More details at the `#haskell` home page: http://haskell.org/haskellwiki/IRC_channel

1.3 Planet Haskell

Report by: Antti-Juhani Kaijanaho
Status: active

Planet Haskell is an aggregator of Haskell people's blogs and other Haskell-related news sites. As of mid-October content from 29 blogs and other sites is being republished in a common format.

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

A blog is eligible to Planet if it is being written by somebody who is active in the Haskell community, or by a Haskell celebrity; also eligible are blogs that discuss Haskell-related matters frequently, and blogs that are dedicated to a Haskell topic (such as a software project written in Haskell). Note that at least one of these conditions must apply, and virtually no blog satisfies them all. However, blogs will not be added to Planet without the blog author's consent.

To get a blog added, email Antti-Juhani Kaijanaho ⟨antti-juhani@kaijanaho.fi⟩ and provide evidence that the blog author consents to this (easiest is to get the author send the email, but any credible method suffices).

Planet is hosted by Galois Connections, Inc. (→

7.1.2) as a service to the community. The Planet maintainer is not affiliated with them.

Further reading

http://planet.haskell.org/

1.4 Haskell Weekly News

Report by:	Don Stewart

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

It is published in html form on The Haskell Sequence, via mail on the Haskell mailing list, on Planet Haskell (→ 1.3), and via RSS. Headlines are published on haskell.org (→ 1.1). The Haskell Weekly News is also available in Spanish translation.

Further reading

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

1.5 Books and tutorials

1.5.1 "Hitchhickers Guide to Haskell" tutorial

Report by:	Dmitry Astapov
Status:	work in progress

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

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

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

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

Further reading

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

1.5.2 New textbook – Programming in Haskell

Report by:	Graham Hutton

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

Features:

- Powerpoint slides for each chapter freely available for instructors and students from the book's website;
- Solutions to exercises and examination questions (with solutions) available to instructors;
- All the code in the book is fully compliant with the latest release of Haskell, and can be downloaded from the web;
- Can be used with courses, or as a stand-along text for self-learning.

Publication details:

- To be published by Cambridge University Press in December 2006.

Further information:

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

1.5.3 Haskell Wikibook (was: Haskell Tutorial Wikibook)

Report by:	Eric Kow
Status:	active development

The Haskell wikibook is an attempt to build a community textbook that is at once free (in cost and remixability), comprehensive and cohesive.

Since the last report, we have overhauled the front page and first few three chapters, and added chapters on advanced topics such as laziness and existential types. We have also been importing a very large amount of content from outside sources, selected pieces of Haskell wiki for starters, as well as the entirety of the excellent tutorials *Write Yourself a Scheme in 48 Hours*

by Johnathan Tang and *Yet Another Haskell Tutorial* by Hal Daumé III. Thanks to all authors, outsiders and wikibook natives alike. As a result of your generous donations, we now have enough content to meet the basic needs for our readers.

Our main focus will thus be on cleaning up what we have and merging it all into a single textbook.

Further reading

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

1.5.4 Haskell Tutorials in Portuguese

Report by:	Diego Navarro (syntaxfree on `#haskell`)
Status:	published online, open to suggestions, translation to english pending

Two weights, two measures

"Two weights, two measures" is a Haskell tutorial focusing on the construction of a very simple DSEL for a fictional prison system exploiting the structure of the Either type (with a few proposed extensions). Its target audience is beginning programmers. The tutorial aims to explore the first steps of how closures/combinators/higher-order functions can be used to define domain specific languages for simple algebraic structures. It's currently available only in portuguese, but it should be translated at some point.

The full text can be found at the URL below.

An introduction to Haskell with autophagic snakes

"An introduction to Haskell with autophagic snakes" is a Haskell tutorial focusing on the exploration of corecursive sequences using infinite lists in Haskell. Its target audience is beginning programmers. The tutorial aims to exempllify lazy evaluation and simple combinators to abstract repetitive structures in the corecursive definitions of sequences. It's currently available only in portuguese, but it should be translated at some point.

The full text can be found at the URL below.

Further reading

- http://www.navarro.mus.br/diego/blog/2006/09/13/tutorial-dois-pesos-duas-medidas/
- http://www.navarro.mus.br/diego/blog/2005/10/20/uma-introducao-ao-haskell-usando-cobras-autofagicas/

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

Report by:	Richard A. Frost

The survey "Realization of Natural-Language Interfaces Using Lazy Functional Programming" is scheduled to be published in ACM Computing Surveys in December 2006. If I have missed any relevant publications, please contact me at rfrost@cogeco.ca. It may be possible to add references before the survey goes to print. If not, I shall put new references on a web page which I am creating to keep the survey up-to-date with future work.

Further reading

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

2 Implementations

2.1 The Glasgow Haskell Compiler

Report by: Simon Peyton-Jones *et al.*

GHC is in good shape. We have no good way to measure how many GHC users there are but if the traffic on the GHC mailing lists is anything to go by, the numbers are increasing quite rapidly. Indeed, GHC was rapidly becoming a success-disaster, so that we (Simon & Simon) were becoming swamped in GHC-related mail. Happily, Microsoft Research has agreed to fund a full-time support engineer, in the form of Ian Lynagh (Igloo), who has already made a huge difference.

A highlight of the last six months was the *GHC Hackathon*, which we ran a immediately before ICFP in Portland, with wonderful support from Galois and Portland State University. Forty-plus people showed up to have GHC's innards inflicted on them, and appeared unharmed by the experience.

A significant outcome is that we have written a great deal of Wiki material about GHC's implementation (the "commentary") and about how to build and modify GHC (the "building guide"). Documents with these titles were available before but had become rather out of date. These new, up-to-date documents live on the GHC developer's Wiki. We urge you to read and improve them: http://hackage.haskell.org/trac/ghc/wiki (near the bottom).

We (finally) released *GHC 6.6* in October 2006. To get GHC 6.6, go to the Download page (http://www.haskell.org/ghc/download_ghc_66.html). There was an extended period of release-candidate testing, so we fondly hope that this will be a relatively stable release. There are many improvements, all listed in the Release notes http://haskell.org/ghc/docs/6.6/html/users_guide/release-6-6.html. The most important new features include:

- Now GHC can execute several Haskell threads simultaneously on different cpus/cores
- `ByteString` (→ 4.6.3) type for fast and memory-efficent string manipulations
- Unicode source files
- Further generalisation of newtype deriving
- Bang patterns to declare function arguments as strict
- Impredicative polymorphism
- Lastly, we finally bit the bullet and lifted the restriction that every module in a Haskell program must have a distinct name. Why? Because it's non-modular: two packages from different authors could accidentally collide. This change is in GHC 6.6; there are some remaining open choices dicussed at http://hackage.haskell.org/trac/ghc/wiki/GhcPackages.

Life still goes on and there is current development version (HEAD), that will ultimately become GHC 6.8. You can find binary snapshots at the download page http://www.haskell.org/ghc/dist/current/dist/ or build from sources available via the darcs repository (http://darcs.haskell.org/ghc/). This version already includes significant new features:

- We have completely replaced GHC's intermediate language with System FC(X), an extension of System F with explicit equality witnesses. This enables GHC to support GADTs and associated types, with two new simple but powerful mechanisms. The paper is at http://research.microsoft.com/~simonpj/papers/ext-f/. Much of the conversion work was done by Kevin Donnelly, while he was on an internship at Microsoft.
- Manuel Chakravarty has implemented *type-indexed data types*, a modest generalisation of the *associated data types* of our POPL'05 paper http://research.microsoft.com/~simonpj/papers/assoc-types/. The implementation is in the HEAD and is ready to be tried out; details are at http://haskell.org/haskellwiki/GHC/Indexed_types. Still to come are some bits around the edges on `deriving` and some small syntactic generalisations.
- Tim Harris added support for *invariants* to GHC's Software Transactional Memory (STM) implementation. Paper here: http://research.microsoft.com/~simonpj/papers/stm/.
- Björn Bringert (a GHC Hackathon graduate) implemented *standalone deriving*, which allows you to write a `deriving` declaration anywhere, rather than only where the data type is declared. Details of the syntax have not yet quite settled. See also http://www.haskell.org/pipermail/haskell-prime/2006-October/001725.html.
- Andy Gill implemented the Haskell Program Coverage (→ 5.4.1) (http://haskell.org/haskellwiki/GHC/HPC) option (`-fhpc`) for GHC, which is solid enough to be used to test coverage in GHC itself. (It turns out that the GHC testsuite gives remarkably good coverage over GHC already.)

If you want to know today's state-of-the-art, you should check the GHC 6.8 status page at http://haskell.org/haskellwiki/GHC/6.8. At this moment we are working on the following features which are planned to be included in GHC 6.8 in next few months:

- Roman Leshchinskiy has been hard at work developing libraries that support *data-parallel computation* in GHC. It's not quite ready for public consumption but you can peek at what is going on by looking at the Haskell Wiki: http://haskell.org/haskellwiki/GHC/Data_Parallel_Haskell. Background material here: http://www.cse.unsw.edu.au/~chak/papers/CKLP01.html. We hope to release a first iteration of our data-parallel extensions before Christmas.

- At the moment GHC's *garbage collector* is single-threaded, even when GHC is running on a multiprocessor. Roshan James spent the summer at Microsoft on an internship, implementing a multi-threaded GC (http://hackage.haskell.org/trac/ghc/wiki/MotivationForParallelization). We need to do a bit more work, but with a bit of luck we'll push a parallel garbage collector into the HEAD before Christmas.

- Simon PJ is determined to finally implement *implication constraints*, which are the key to fixing the interaction between GADTs and type classes. GHC's users have been very polite about this collection of bugs, but they should really be fixed. Implication constraints are described by Martin Sulzmann: http://www.comp.nus.edu.sg/~sulzmann/publications/tr-eadt.ps.gz.

- Once the last bits of indexed data types are done, Manuel will be tackling indexed type synonyms (aka type functions), which are considerably trickier, at least so far as type inference is concerned.

2.2 Hugs

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

The September 2006 release of Hugs fixes a few bugs found in the previous release, and updates the libraries to approximately match those of GHC 6.6, which was about to release at the time. The Windows build is now largely automated, thanks to Neil Mitchell, so it is easier to produce more frequent releases.

As with the previous release, the source distribution is available in two forms: a huge omnibus bundle containing the Hugs programs and lots of useful libraries, or a minimal bundle, with most of the libraries hived off as separate Cabal packages. We hope that more library packages will be released independently, so that Hugs will become less reliant on development snapshots.

Obsolete non-hierarchical libraries will be removed in the next major release.

As ever, volunteers are welcome.

2.3 nhc98

Report by:	Malcolm Wallace
Status:	stable, maintained

nhc98 is a small, easy to install, compiler for Haskell'98. Despite rumours to the contrary, nhc98 is still very much alive and working, although it does not see any new development these days. The current public release is version 1.18, with a new release expected soon for compatibility with ghc-6.6 and the re-arranged hierarchical libraries. We recently moved over to a darcs repo for maintenance.

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

Further reading

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

2.4 yhc

Report by:	Neil Mitchell

The York Haskell Compiler (yhc) is a fork of the nhc98 (→ 2.3) compiler, with goals such as increased portability, platform independent bytecode, integrated Hat support and generally being a cleaner code base to work with. Yhc now compiles and runs almost all Haskell 98 programs, has basic FFI support – the main thing missing is haskell.org base libraries, which is being worked on.

Since that last HCAR a lot of work has been put in place on the infrastructure of the project – we now have a new build system, nightly builds, automated testing, snapshot releases, wiki documentation, a haskell.org

darcs repository. There has also been a focus on creating libraries to allow programmers to reuse some of the work done by Yhc – in particular Yhc.ByteCode and Yhc.Core.

Going forward our focus is to support the haskell.org base libraries, get full Cabal support, enhance our Yhc.* libraries, refactor everything and to support Haskell' fully.

Further reading

- Homepage:
  http://www.haskell.org/haskellwiki/Yhc
- Darcs repository:
  http://darcs.haskell.org/yhc

3 Language

3.1 Variations of Haskell

3.1.1 Haskell on handheld devices

Report by:	Anthony Sloane
Participants:	Michael Olney
Status:	unreleased

The project at Macquarie University (→ 7.3.5) to run Haskell on handheld devices based on Palm OS is close to producing a working implementation for experimentation. Our port of the yhc (→ 2.4) runtime is now running small examples on simulators and real devices. We are currently testing with larger GUI-based programs. We expect to make a public alpha release sometime in the (southern) summer.

3.1.2 Pivotal: Visual Interactive Programming

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

Pivotal 0.025 is a very early prototype of a Vital-like environment for Haskell. Unlike Vital, however, Pivotal is implemented entirely in Haskell. The implementation is based on the use of the hs-plugins library (→ 4.4.1) to allow dynamic compilation and evaluation of Haskell expressions together with the gtk2hs library (→ 4.8.2) for implementing the GUI.

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

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

Further reading

- Pivotal: http://www.cs.kent.ac.uk/projects/pivotal/
- Vital: http://www.cs.kent.ac.uk/projects/vital/

3.1.3 Camila

Report by:	Jácome Cunha and Joost Visser

The Camila project explores how concepts from the VDM++ specification language and the functional programming language Haskell can be combined. On one hand, it includes experiments of expressing VDM's data types (e.g. maps, sets, sequences), data type invariants, pre- and post-conditions, and such within the Haskell language. On the other hand, it includes the translation of VDM specifications into Haskell programs. Moreover, the use of the OOHaskell library (→ 4.6.6) allows the definition of classes and objects and enables important features such as inheritance. In the near future, support for parallelism and automatic translation of VDM++ specifications into Haskell will be added to the libraries.

Camila goes beyond VDM++ and has support for modelling software components. The work done until now in this field is concerned with rendering and prototyping (coalgebraic models of) software components in Camila. To encourage the use of this technology we have developed a tool to generate components from Camila specifications. The advantage of component based development is that it makes possible to construct complex software from simple pre-existing building blocks. So we have also animated an algebra of components to compose them in several ways. Finally a way to animate components was also implemented.

Two implementation strategies were devised: one in terms of a direct encoding in "plain" Haskell, another resorting to type-level programming techniques, the latter offered interesting particularities.

Further reading

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

3.1.4 HASP

Report by:	Lemmih
Status:	active

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

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

Some of the features implemented in HASP will be ported back into the main HSP tree. However, experi-

mental features like byte code generation via the GHC api will most likely stay in HASP.

Further reading

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

3.2 Non-sequential Programming

3.2.1 GpH – Glasgow Parallel Haskell

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

Status

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

System Evaluation and Enhancement

- A major revision of the parallel runtime environment for GHC 6.5 is currently under development. Support for the parallel language Eden (→ 3.2.3) exists and is currently being tested. Support for the parallel language GpH is currently being added to this version of the runtime environment.

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

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

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

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

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

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

GpH Applications

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

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

Implementations

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

- The focus of the development has switched to the version based on GHC 6.5, and we plan to make an early prototype available from the GpH web site later this year.

- The stable branch (GUM-4.06, based on GHC-4.06) is available for RedHat-based Linux machines. The stable branch is available from the GHC CVS repository via tag gum-4-06. 'item A current unstable branch (GUM-5.02, based on GHC-5.02) is available on request.

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

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

Further reading

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

Contact

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

3.2.2 GdH – Glasgow Distributed Haskell

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

GdH supports distributed stateful interactions on multiple locations. It is a conservative extension of both Concurrent Haskell and GpH (→ 3.2.1), enabling the distribution of the stateful IO threads of the former on the multiple locations of the latter. The programming model includes forking stateful threads on remote locations, explicit communication over channels, and distributed exception handling.

Status

An alpha-release of the GdH implementation is available on request ⟨gph@macs.hw.ac.uk⟩. It shares substantial components of the GUM implementation of GpH (Glasgow parallel Haskell) (→ 3.2.1).

Applications and Evaluation

- EPSRC project *High Level Techniques for Distributed Telecommunications Software* (GR/R88137) has recently been completed (February 2006). The project was collaboration between Heriot-Watt University and Motorola UK Research Labs, and amongst other activities evaluated GdH and Erlang as technologies for distributed telecoms software in comparison to C++/CORBA. Previous publications appear on the project page, and some recent results are given in the papers below. The latter paper compares Erlang, GdH and C++ for engineering a medium-scale (14K lines of C++, 4K lines of Erlang, and 0.5K lines of GdH) telecoms component.

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

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

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

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

Further reading

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

3.2.3 Eden

Report by:	Rita Loogen

Description

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

in Madrid: Ricardo Peña, Yolanda Ortega-Mallén, Mercedes Hidalgo, Fernando Rubio, Clara Segura, Alberto Verdejo

in Marburg: Rita Loogen, Jost Berthold, Steffen Priebe, Björn Struckmeier

Eden extends Haskell with a small set of syntactic constructs for explicit process specification and creation. While providing enough control to implement parallel algorithms efficiently, it frees the programmer from the tedious task of managing low-level details by introducing automatic communication (via head-strict lazy lists), synchronisation, and process handling.

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

Survey and standard reference

Rita Loogen, Yolanda Ortega-Mallén and Ricardo Peña: *Parallel Functional Programming in Eden*, Journal of Functional Programming 15(3), 2005, pages 431–475.

Implementation

A major revision of the parallel Eden runtime environment for GHC 6.5 is available. Support for Glasgow parallel Haskell (GpH) is currently being added to this version of the runtime environment. It is planned for the future to maintain a common parallel runtime environment for Eden, GpH and other parallel Haskells.

Recent and Forthcoming Publications

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

- Jost Berthold, Rita Loogen: *Parallel Coordination Made Explicit in a Functional Setting*, IFL, Budapest, September 2006.
- Mercedes Hidalgo-Herrero, Yolanda Ortega-Mallén, Fernando Rubio: *Analyzing the influence of mixed evaluation on the performance of Eden skeletons*, Parallel Computing 32 (2006) 523–538.
- Mercedes Hidalgo-Herrero, Alberto Verdejo, Yolanda Ortega-Mallén: *Using Maude and its strategies for defining a framework for analyzing Eden semantics*, WRS 06 (6th International Workshop on Reduction Strategies in Rewriting and Programming), Aachen 2006.
- Steffen Priebe: *Dynamic Task Generation and Transformation within a Nestable Workpool Skeleton*, Euro-Par 2006, LNCS 4128, Springer 2006.
- Björn Struckmeier: *Implementing a tool for visualising and analysing parallel program runs in Haskell*, Diploma Thesis, Philipps-Universität Marburg, September 2006 (in German).

Further reading

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

3.3 Type System/Program Analysis

3.3.1 Epigram

Report by:	Conor McBride and Wouter Swierstra

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

Motivation

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

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

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

Current Status

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

The Epigram system is also being used successfully by Thorsten Altenkirch, and more recently Conor McBride, in an undergraduate course on Computer Aided Formal Reasoning for two years http://www.e-pig.org/darcs/g5bcfr/. Several final year students have successfully completed projects that involved both new applications of and useful contributions to Epigram.

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

We have started writing a stand-alone editor for Epigram using Gtk2Hs (→ 4.8.2). Thanks to a most helpful visit from Duncan Coutts and Axel Simon, two leading Gtk2Hs developers, we now have the beginnings of a structure editor for Epigram 2. For the moment, we are also looking into a cheap terminal front-end.

There has also been steady progress on Epigram 2 itself. Most of the recent progress has been on the type theoretic basis underpinning Epigram. A new representation of the core syntax has been designed to facilitate bidirectional type checking. The semantics of individual terms are glued to their syntactical representation. We have started implementing *observational equality*, combining the benefits of both intensional and exten-

sional notions of equality. The lion's share of the core theory has already been implemented, but there is still plenty of work to do.

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

Epigram source code and related research papers can be found on the web at http://www.e-pig.org and its community of experimental users communicate via the mailing list ⟨epigram@durham.ac.uk⟩. The current implementation is naive in design and slow in practice, but it is adequate to exhibit small examples of Epigram's possibilities. The new implementation will be much less rudimentary. At the moment, there is direct low-level interface to the state of the proof state called Ecce. Its documentation, together with other Epigram 2 design documents, can be found at http://www.e-pig.org/epilogue/.

3.3.2 Chameleon project

Report by:	Martin Sulzmann

Chameleon is a Haskell style language which integrates sophisticated reasoning capabilities into a programming language via its CHR programmable type system. Thus, we can program novel type system applications in terms of CHRs which previously required special-purpose systems.

Latest developments

A new version of Chameleon including examples and documentation is available via http://taichi.ddns.comp.nus.edu.sg/taichiwiki/ChameleonHomePage

3.3.3 XHaskell project

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

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

Latest developments

A new version of XHaskell including examples and documentation is available via http://taichi.ddns.comp.nus.edu.sg/taichiwiki/XhaskellHomePage

3.3.4 ADOM: Agent Domain of Monads

Report by:	Martin Sulzmann
Participants:	Edmund S. L. Lam and Martin Sulzmann

ADOM is an agent-oriented extension of Haskell with a unique approach to the implementation of cognitive Belief-Desire-Intention (BDI) agents. In ADOM, agent reasoning operations are viewed as monadic computations. Agent reasoning operations can be stratified: Low-level reasoning operations involve the agents beliefs and actions whereas high-level reasoning operations involve the agents goals and plans. Monads allow us to compose various levels of reasoning together, while maintaining clear and distinct separation between the different levels. ADOM can be used directly as an agent-oriented domain specific language, or used to build more higher level BDI agent abstractions on top of it (eg. AgentSpeak, 3APL). ADOM also introduces the use of Constraint Handling Rules (CHR), embedded with Haskell, to directly model the agent's belief of its dynamically changing domain (world) and it's actions which invoke change to it's domain. The key advantage of our approach are:

- CHRs provides a clear and concise representation and implementation of dynamically changing agent beliefs and actions.
- Stratifying the various levels of agent cognitive reasoning by monads, maintains a distinct separation between different reasoning computations and their responsibilities. We can also preserve certain desirable properties possessed by each level of computations. For example, CHR notion of observable confluence.
- Monadic computations can be composed to form more complex computations, hence ADOM can be easily extended with more complex functionalities. For example, we can build higher level monadic computations that implements other BDI frameworks, like agentspeak or 3APL.

Further reading

More information on ADOM can be found here http://taichi.ddns.comp.nus.edu.sg/taichiwiki/ADOMHomePage

3.3.5 EHC, 'Essential Haskell' Compiler

Report by:	Atze Dijkstra
Participants:	Atze Dijkstra, Jeroen Fokker, Arie Middelkoop, Doaitse Swierstra
Status:	active development

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

For its description an Attribute Grammar system (AG) (→ 4.3.1) is used as well as other formalisms allowing compact notation like parser combinators. For the description of type rules, and the generation of an AG implementation for those type rules, we use the Ruler system (→ 5.5.2) (included in the EHC project).

The EHC project also tackles other issues:

- In order to avoid overwhelming the innocent reader, the description of the compiler is organised as a series of increasingly complex steps. Each step corresponds to a Haskell subset which itself is an extension of the previous step. The first step starts with the essentials, namely typed lambda calculus.
- Each step corresponds to an actual, that is, an executable compiler. Each of these compilers is a compiler in its own right so experimenting can be done in isolation of additional complexity introduced in later steps.
- The description of the compiler uses code fragments which are retrieved from the source code of the compilers. In this way the description and source code are kept synchronized.

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

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

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

Current activities

We are currently working on the following:

- A Haskell98 frontend, supporting most of Haskell98, done by Atze Dijkstra.
- A GRIN (Graph Reduction Intermediate Notation, see below) like backend, which allows experimenting with global program optimization. This is done by Jeroen Fokker.
- Arie Middelkoop will continue with the development of the Ruler system (→ 5.5.2).

Further reading

- Homepage:
  http://www.cs.uu.nl/groups/ST/Ehc/WebHome
- Attribute grammar system:
  http://www.cs.uu.nl/wiki/HUT/AttributeGrammarSystem
- Parser combinators:
  http://www.cs.uu.nl/wiki/HUT/ParserCombinators
- GRIN:
  Urban Boquist, Code Optimisation Techniques for Lazy Functional Languages, PhD Thesis, Chalmers University of Technology 1999
  http://www.cs.chalmers.se/~boquist/phd/index.html

3.3.6 Uniqueness Typing in EHC

Report by:	Arie Middelkoop
Participants:	Arie Middelkoop, Jurriaan Hage
Status:	Prototype finished

Uniqueness typing is a type system feature of the functional programming language Clean to identify unique values. The space these values occupy can be recycled directly after their only use, thus enabling a form of static garbage collection that greatly improves the efficiency of functional programs. Our goal is to take this idea, and use it to produce more efficient Haskell code.

This project consists of two parts: an analysis to determine which values are unique (front-end), and a code specializer that uses the analysis results to optimize memory management (back-end). We did focus on the front-end part and implemented a prototype using the Essential Haskell (→ 3.3.5) project as a research vehicle. Code generation is ongoing work of the Essential Haskell project, and we intent to integrate the results of the uniqueness analysis in a later phase.

Our uniqueness analyzer works as follows. Each type constructor of a well-typed program is annotated with a fresh identifier called the uniqueness annotation. From the structure of the AST, we generate a bunch of constraints between these annotations. Solving the constraints gives a local reference count (taking the current slice of the program into account) and global reference count (taking the whole program into account) of each annotation. The global reference count is constructed from the local reference counts and serves as an approximation of an upper bound to the actual usage of a value. (Sub)values that end up with an upper bound are considered unique, others are shared.

Further reading

- Master's thesis:
  http://abaris.zoo.cs.uu.nl:8080/wiki/pub/Top/Publications/uniqueness.pdf

- Sources:
  https://svn.cs.uu.nl:12443/repos/EHC/branches/uniqueness/EHC/
- EH project page:
  http://www.cs.uu.nl/groups/ST/Ehc/WebHome

3.3.7 Object-Oriented Haskell

Report by:	Glenn Strong
Status:	ongoing

A set of type and other extensions to a Haskell-derived language to support the general notion of Object-Oriented programming. An interpreter is under construction to provide a programming environment. No public release is currently available as the system is not yet usable.

3.4 IO

3.4.1 Formal Aspects of Pure Functional I/O

Report by:	Andrew Butterfield
Participants:	Andrew Butterfield, Glenn Strong, Malcolm Dowse
Status:	ongoing

We are particularly interested in formal models of the external effects of I/O in pure lazy functional languages. The emphasis is on reasoning about how programs affect their environment, rather than the issue of which programs have identical I/O behaviour.

Further reading

BS01 Andrew Butterfield and Glenn Strong, "Proving correctness of programs with I/O — a paradigm comparison", in Thomas Arts and Markus Mohnen, editors, *Proceedings of the 13th International Workshop, IFL2001*, LNCS 2312, pages 72–87, 2001.

BDS02 Malcolm Dowse, Glenn Strong, and Andrew Butterfield, "Proving make correct — I/O proofs in Haskell and Clean", in Ricardo Peña and Thomas Arts, editors, *Proceedings of IFL 2002*, LNCS 2670, pages 68–83, 2002

BDE04 Malcolm Dowse, Andrew Butterfield, and Marko van Eekelen, "Reasoning about deterministic concurrent functional i/o", in Clemens Grelck, Frank Huch, and Phil Trinder, editors, *IFL'04 - Revised Papers*, LNCS 3474, 2005.

BD06 Malcolm Dowse, Andrew Butterfield, "Modelling Deterministic Concurrent I/O", in Julia Lawall, editor, ICFP 2006, Portland, September 18–20, 2006.

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. The techniques behind some of these ideas are given in a separate subsection. In *Comparing approaches to generic programming in Haskell* (in the lecture notes of the Spring School on Datatype-Generic Programming 2006, held in Nottingham, April 2006, to appear in LNCS), Ralf Hinze, Johan Jeuring and Andres Löh compare 8 different approaches to generic programming in Haskell, both lightweight approaches and language extensions. Most of the approaches discussed in this and previous versions of the Communities report are addressed. In the same set of lecture notes, Jeremy Gibbons discusses the various interpretations of the word 'generic'.

Preprocessors

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

Languages

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

Generic functions for data type traversals can (almost) be written in Haskell itself (using many of the extensions of Haskell provided by GHC), as shown by Ralf Lämmel and Simon Peyton Jones in the 'Scrap your boilerplate' (SYB) approach (http://www.cs.vu.nl/boilerplate/). The SYB approach to generic programming in Haskell has been further elaborated in the recently published (in FLOPS '06) paper *"Scrap Your Boilerplate" Reloaded* and *"Scrap Your Boilerplate" Revolutions* (to appear in MPC'06). In these pa-

pers Ralf Hinze, Andres Löh, and Bruno Oliveira show, amongst others, how by viewing the SYB approach in a particular way, the choice of basic operators becomes obvious.

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

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

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

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

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

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

Techniques

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

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

Further reading

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

4 Libraries

4.1 Packaging and Distribution

4.1.1 Core

Report by:	Bulat Ziganshin
Status:	experimental

Thanks to Cabal, we can now easily upgrade any installed library to a new version. There is only one exception: the Base library is closely tied to compiler internals, so you cannot use the Base library shipped with GHC 6.4 in GHC 6.6 and vice versa.

The Core library is a project of dividing the Base library into two parts – a small compiler-specific one (the Core library proper) and the rest – a new, compiler-independent Base library that uses only services provided by the Core lib.

Then, any version of the Base library can be used with any version of the Core library, i.e. with any compiler. Moreover, it means that the Base library will become available for the new compilers, like yhc (→ 2.4) and jhc – this will require adding to the Core lib only a small amount of code implementing low-level compiler-specific functionality.

The Core library consists of directories GhcCore, HugsCore ... implementing compiler-specific functionality and Core directory providing common interface to this functionality, so that external libs should import only Core.* modules in order to be compiler-independent.

In practice, the implementation of the Core lib became a refactoring of the GHC.* modules by splitting them into GHC-specific and compiler-independent parts. Adding implementations of compiler-specific parts for other compilers will allow us to compile the refactored Base library with any compiler, including old versions of GHC. At this moment, the following modules were succesfully refactored: GHC.Arr, GHC.Base, GHC.Enum, GHC.Float, GHC.List, GHC.Num, GHC.Real, GHC.Show, GHC.ST, GHC.STRef; the next step is to refactor IO functionality.

Further reading

- Documentation page: http://haskell.org/haskellwiki/Library/Core
- Download: http://www.haskell.org/library/Core.tar.gz

Contact

⟨Bulat.Ziganshin@gmail.com⟩

4.2 General libraries

4.2.1 PFP – Probabilistic Functional Programming Library for Haskell

Report by:	Martin Erwig
Status:	mostly stable, not maintained

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

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

The library was developed as part of a simulation project with biologists and genome researchers. We originally had planned to apply the library to more examples in this area, however, the student working in this area has left, so this project is currently in limbo.

No changes since the last report.

Further reading

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

4.2.2 Hmm: Haskell Metamath module

Report by:	Marnix Klooster
Status:	Hmm 0.2 released, slow-paced development

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

Metamath (http://metamath.org) was conceived and almost completely implemented by Norman Megill. It a project for formalizing mathematics, a file format for specifying machine-checkable proofs, and a program for generating and verifying this file format. Already more than 7500 proofs have been verified from the axioms of set theory.

Version 0.2 of Hmm has been released on October 28th, 2005.

The development version can be found at http://www.solcon.nl/mklooster/repos/hmm/. This is a darcs repository (→ 6.4).

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

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

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

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

Further reading

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

4.2.3 GSLHaskell

Report by:	Alberto Ruiz
Status:	active development

GSLHaskell is a high level functional interface to linear algebra and other numerical computations internally implemented using GSL and LAPACK. The goal is to achieve the functionality and performance of GNU-Octave or similar system s. The library is still very incomplete, but it has already been useful in a few elementary pattern recognition and computer vision applications.

Recent developments include a simplification of the interface and improved linear algebra based on LAPACK.

Further reading

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

4.2.4 An Index Aware Linear Algebra Library

Report by:	Frederik Eaton
Status:	unstable; actively maintained

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

Currently the library is in a "proof-of-concept" state. The interface has an example implementation using Arrays, but ultimately it should be primarily used with a fast external linear algebra package such as ATLAS. I would like to see it become part of Alberto Ruiz's GSL library (→ 4.2.3), which can be used with ATLAS, and he has expressed an interest in adopting it. That is why I haven't given it a real name yet.

The original announcement is here:

Further reading

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

4.2.5 Ivor

Report by:	Edwin Brady
Status:	active development

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

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

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

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

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

Further reading

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

4.2.6 Haskell Rules: Embedding Rule Systems in Haskell

Report by:	Martin Erwig
Status:	mostly stable, not maintained

Haskell Rules is a domain-specific embedded language that allows semantic rules to be expressed as Haskell functions. This DSEL provides logical variables, unification, substitution, non-determinism, and backtracking. It also allows Haskell functions to be lifted to operate on logical variables. These functions are automatically delayed so that the substitutions can be applied. The rule DSEL allows various kinds of logical embedding, for example, including logical variables within a data structure or wrapping a data structure with a logical wrapper.

Further reading

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

4.3 Parsing and transforming

4.3.1 Utrecht Parsing Library and Attribute Grammar System

Report by:	Doaitse Swierstra
Status:	Released as cabal packages

The Utrecht attribute grammar system has been extended:

- the attribute flow analysis has been completely implemented by Joost Verhoog, and it is now possible to generate visit-function based evaluators, which are much faster and use less space. We assume that such functions are strict in all their arguments, and generate the appropriate |'seq'| calls to make the GHC aware of this. As a result also |case|'s are generated instead on |let|'s wherever possible.
- the base system has been extended by Jeroen Fokker with wildcard notations for designating groups of productions, attributes etc.

We are in the process of integerating these developments and making the software again available through the Haskell Utrecht Tools page. (http://www.cs.uu.nl/wiki/HUT/WebHome).

Some small changes were made to the cabal files in order to make installation under GHC 6.6 run.

4.3.2 Left-Recursive Parser Combinators

Report by:	Richard A. Frost
Participants:	Rahmatullah Hafiz, Paul Callaghan
Status:	Pre-release

Existing parser combinators cannot accommodate left-recursive grammars. In some applications, this shortcoming requires grammars to be rewritten to non-left-recursive form which may hinder definition of the associated semantic functions. In applications that involve ambiguous pattern-matching, such as NLP, the rewriting to non-left-recursive form may result in loss of parses.

In our project, we have developed combinators which accommodate ambiguity and left-recursion (both direct and indirect) in polynomial time, and which generate polynomial-sized representations of the exponential number of parse trees corresponding to highly-ambiguous input. The compact representations are similar to those generated by Tomita's algorithm.

Polynomial complexity for ambiguous grammars is achieved through memoization of fully-backtracking combinators. Systematic memoization is implemented using monads. Direct left-recursion is accommodated by storing additional data in the memotable which is

used to curtail recursive descent when no parse is possible. Indirect left recursion is accommodated by use of the context in which results are created and the context in which they are subsequently considered for re-use.

We have implemented our approach in Haskell, and are in the process of optimizing the code and preparing it for release in December of 2006.

Further reading

A technical report with definitions, proofs of termination and complexity, and reference to publications, is available at: http://cs.uwindsor.ca/~richard/GPC/TECH_REPORT_06_022.pdf

4.3.3 RecLib – A Recursion and Traversal Library for Haskell

Report by:	Martin Erwig
Status:	mostly stable, not maintained

The Recursion Library for Haskell provides a rich set of generic traversal strategies to facilitate the flexible specification of generic term traversals. The underlying mechanism is the Scrap Your Boilerplate (SYB) approach. Most of the strategies that are used to implement recursion operators are taken from Stratego.

The library is divided into two layers. The high-level layer defines a universal traverse function that can be parameterized by five aspects of a traversal.

The low-level layer provides a set of primitives that can be used for defining more traversal strategies not covered in the library. Two fixpoint strategies innter-most and outermost are defined to demonstrate the usage of the primitives. The design and implementation of the library is explained in a paper listed on the project web page.

Further reading

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

4.4 System

4.4.1 hs-plugins

Report by:	Don Stewart
Status:	active development

hs-plugins is a library for dynamic loading and runtime compilation of Haskell modules, for Haskell and foreign language applications. It can be used to implement application plugins, hot swapping of modules in running applications, runtime evaluation of Haskell, and enables the use of Haskell as an application extension language. Version 1.0rc1 has been released.

Work to port hs-plugins to GHC 6.6 is underway.

Further reading

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

4.4.2 time (was: Package "time")

Report by:	Ashley Yakeley
Status:	stable

The "time" package replaces the old System.Time module for handling time. It is included in the current GHC distribution.

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

The source is in the darcs (→ 6.4) repository "time" in the current standard libraries, and is built by the GHC library build process. The documentation could benefit from the addition of use examples.

Further reading

http://semantic.org/TimeLib/

4.4.3 The libpcap Binding

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

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

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

Ther have been no changes since April 2006.

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

All contributions are welcome.

4.4.4 Streams

Report by:	Bulat Ziganshin
Status:	beta, actively developed

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

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

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

The new version 0.2 adds support for memory-mapped files, files >4GB on Windows, ByteString I/O, full backward compatibility with the NewBinary library (both byte-aligned and bit-aligned modes), more orthogonal serialization API, serialization from/to memory buffer, and even better speed. Sorry, it was never documented

The upcoming version 0.3 will provide automatic buffer deallocation using ForeignPtrs, serialization from/to ByteStrings, full backward compatibility with Handle-base I/O and, hopefully, full documentation for all its features.

Further reading

- Documentation page: http://haskell.org/haskellwiki/Library/Streams
- Download: http://www.haskell.org/library/Streams.tar.gz http://www.haskell.org/library/StreamsBeta.tar.gz

Contact

⟨Bulat.Ziganshin@gmail.com⟩

4.4.5 System.FilePath

Report by:	Neil Mitchell

System.FilePath is a library for manipulating FilePath's in Haskell programs. This library is Posix (Linux) and Windows capable – just import System.FilePath and it will pick the right one. It is written in Haskell 98 + Hierarchical Modules. There are features to manipulate the extension, filename, directory structure etc. of a FilePath.

This module has received significant discussion on the Haskell mailing lists, and going forward I hope that it can be incorporated into the Haskell base libraries.

Further reading

http://www-users.cs.york.ac.uk/~ndm/projects/libraries.php#filepath

4.4.6 hinotify

Report by:	Lennart Kolmodin
Status:	alive

hinotify is a simple Haskell wrapper for the Linux kernel's inotify mechanism. inotify allows applications to watch file changes since Linux kernel 2.6.13. You can for example use it to do a proper locking procedure on a set of files, or keep your application up do date on a directory of files in a fast and clean way.

hinotify is still a very young library and might still be a bit rough around the edges. Next updates will include non-threading support and perhaps a little reworked API.

Further reading

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

4.5 Databases and data storage

4.5.1 CoddFish

Report by:	Alexandra Silva and Joost Visser

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

The library makes essential use of the HList library (→ 4.6.6), which provides arbitrary-length tuples (or heterogeneous lists), and makes extensive use of type-level programming with multi-parameter type classes.

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

Currently, a reimplementation of CoddFish based on GADTs is underway.

Further reading

- Project URL:
  http://wiki.di.uminho.pt/wiki/bin/view/PURe/CoddFish
- Paper: Alexandra Silva and Joost Visser, *Strong Types for Relational Databases (Functional Pearl)*, in Proceedings of Haskell Workshop 2006

4.5.2 Takusen

Report by:	Alistair Bayley
Status:	active development

Takusen is a library for accessing DBMS's. Like HSQL, we support arbitrary SQL statements (currently strings, extensible to anything that can be converted to a string). Takusen's 'unique-selling-point' is 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. We also support invoking database stored procedures with In/Out parameters, and processing of SQL statements that return multiple result sets. Currently we fully support Oracle, Sqlite, and PostgreSQL.

Since the last report we have:

- completed move to darcs repo at
  http://darcs.haskell.org/takusen/
- added support for returning multiple result-sets from queries or stored procedure invocations for Oracle and PostgreSQL
- updated Haddock documentation
- Added COPY-IN (bulk load) to mid-level PostgreSQL interface
- written sample program for the middle-level PostgreSQL interface, which is actually used as simple database interface for other languages e.g. Scheme
- added support for Data.Time (→ 4.4.2). This is the only date-time support we have for the PostgreSQL interface (Oracle and Sqlite still support CalendarTime, FWIW).
- cabalised. Cabal support is a bit patchy at present. It doesn't work with GHC-6.6 out of the box; you need to upgrade Cabal to 1.16.1. If you have GHC-6.4, then you need to install Data.Time first, and this can be a bit of a chore. If you have trouble, contact us and we'll try to help you.

Future plans

- smooth cabal installation with GHC-6.6
- ODBC interface
- MS SQL Server interface (this needs COM support, so we may wait for VisualHaskell to mature a bit before attempting this)

Further reading

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

4.6 Data types and data structures

4.6.1 Standard Collection Libraries

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

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

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

This new package features notably:

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

The collection package is ready for experimental use by the Haskell community. An important difference with other collection frameworks is that this one is intended as an evolution rather that a revolution. It should be easy to migrate code from using Data.Map/Set to the new framework.

Future plans include:

- Add more trie-based data structures;
- Port the class framework to associated types.

Further reading

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

4.6.2 The revamped monad transformer library

Report by:	Iavor Diatchki
Status:	mostly stable

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

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

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

Further reading

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

4.6.3 Data.ByteString

Report by:	Don Stewart
Status:	active development

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

Data.ByteString is written in Haskell98 plus the foreign function interface and cpp. It has been tested succesfully with GHC 6.4 and 6.5, and hugs March 2005.

It has been a period of great activity for Data.ByteString, which is now part of the fptools base libraries, and installed by default with GHC 6.6 and Hugs 2006. The array fusion system has been completely rewritten to use streams, with dramatic speed improvements. This work appears in our recent "Rewriting Haskell Strings" paper.

Further reading

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

4.6.4 Edison

Report by:	Robert Dockins
Status:	active development

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

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

- API Typeclasses updated to use functional dependencies
- Modules rearranged to use the hierarchical module extension
- Documentation move from separate document to inline Haddock comments (→ 5.5.6)
- Source code is now managed in a darcs repository (→ 6.4)
- New Cabal-based build system
- A full suite of unit tests, which covers the entire Edison API, is now included
- Numerous additions to the Associated Collection API
- Several new data structure implementations

Further reading

- The project home page may be found at: http://www.eecs.tufts.edu/~rdocki01/edison.html
- The main darcs repository (→ 6.4) is located at: http://www.eecs.tufts.edu/~rdocki01/edison/

4.6.5 Numeric prelude

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

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

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

Currently supported algebraic structures are

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

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

- complex number, quaternion,
- residue class,
- fraction,
- fixed point arithmetic with respect to arbitrary bases and numbers of fraction digits,
- infinite precision number in an arbitrary positional system as lazy lists of digits supporting also numbers with terminating representations,
- polynomial, power series, Laurent series
- root set of a polynomial,
- numbers equipped with physical units (dynamic checks only).

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

Using the revised system requires hiding some of the standard functions provided by Prelude, which is fortunately supported by GHC (→ 2.1). The library has basic Cabal support and a growing test-suite of QuickCheck tests for the implemented mathematical objects.

Future plans

Collect more Haskell code related to mathematics, e.g. for linear algebra. Study of alternative numeric type class proposals and common computer algebra systems. Ideally each data type resides in a separate module. However this leads to mutual recursive dependencies, which cannot be resolved if type classes are mutually recursive. We start to resolve this by fixing the types of some parameters of type class methods. E.g. power exponents become simply Integer instead of Integral, which has also the advantage of reduced type defaulting.

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

Further reading

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

4.6.6 HList – a library for typed heterogeneous collections

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

HList is a comprehensive, general purpose Haskell library for typed heterogeneous collections including extensible records and variants. 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 lists, elements of heterogeneous lists do not have to have the same type. HList lets the user formulate statically checkable constraints: for example, no two elements of a collection may have the same type (so the elements can be unambiguously indexed by their type).

An immediate application of HLists is the implementation of open, extensible records with first-class, reusable, and compiled-time only labels. We and now others (Alexandra Silva, Joost Visser: PURe.CoddFish project (→ 4.5.1)) have also used HList for type-safe database access in Haskell. HList-based Records

form the basis of OOHaskell http://darcs.haskell.org/OOHaskell. The HList library relies on common extensions of Haskell 98.

We have changed the representation of (extensible and polymorphic) Records. Now, the field label information is purely phantom, that is, compile-time only. At run-time, a record is just a heterogeneous list of field values. We realize records as sequences of field values, where the type of each field is annotated with its (phantom) label. We also present an alternative; it too, represents records as sequences of field values; only now the type of the entire sequence is annotated with the phantom type sequence of the corresponding labels. The latter representation can easily realize 'tables'.

We have added the implementation of extensible polymorphic variants (open unions), as duals of records. We can re-use as much of old code as possible, when adding new alternatives to the variant and extending the functions to the extended variant. We obtain the variant subtyping for free.

The HList repository is now moved to Darcs (→ 6.4): http://darcs.haskell.org/HList

We are working on Cabalizing HList, expanding on the work by Einar Karttunen.

Further reading

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

4.6.7 ArrayRef

Report by:	Bulat Ziganshin
Status:	beta

This is a Hugs (→ 2.2) and GHC (→ 2.1) compatible library for "improved arrays and references" featuring:

- Unboxed references in the IO and ST monads, that supports all simple datatypes and an IORef/STRef-like interface. This replaces the widely used "fast unboxed variables" modules.
- A monad-independent interface to boxed and unboxed references that allows to implement algorithms executable both in the IO and ST monads
- Syntactic sugar for references, mutable arrays and hash tables (`=:`, `+=`, `-=`, `.=`, `val`, `ref`, `uref`)
- Refactored implementation of Data.Array.* modules. Changes include support for dynamic (resizable) arrays and polymorphic unboxed arrays

  (http://www.haskell.org/pipermail/haskell-cafe/2004-July/006400.html),

Further reading

- Documentation page: http://haskell.org/haskellwiki/Library/ArrayRef
- Download: http://www.haskell.org/library/ArrayRef.tar.gz

Contact

⟨Bulat.Ziganshin@gmail.com⟩

4.7 Data processing

4.7.1 HsSyck

Report by:	Audrey Tang
Status:	active development

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

Syck is an extension, written in C, for reading and writing YAML swiftly in popular scripting languages. It is part of core Ruby, and also has bindings for Perl 5, Python, Lua, Cocoa, and Perl 6.

HsSyck provides Data.Yaml.Syck as an interface to YAML structures, using Data.ByteString (→ 4.6.3) for efficient textual data representation. Additionally, we provide a set of DrIFT rules (→ 3.5) to dump and load arbitrary Haskell data types in the YAML format.

Further reading

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

4.7.2 AltBinary

Report by:	Bulat Ziganshin
Status:	beta, actively developed

AltBinary is a part of the Streams library (→ 4.4.4). AltBinary implements binary I/O and serialization facilities. It features:

- Hugs and GHC compatibility
- Lightning speed (3-20 times faster than GHC Binary)
- Classical get/put Binary class interface
- Full backward compatibility with NewBinary lib
- Byte-aligned and bit-aligned, low-endian and big-endian serialization
- Serialization of all widely used types (integral, enums, float, arrays, maps ...)
- UTF8 encoding for strings/chars

- Ability to use TH to derive Binary instance for any type
- Over 50 custom serialization routines (putWord32LE, putMArrayWith ...)
- Ability to serialize data to any Stream what implements vPutByte/vGetByte operations, including support for monads other than IO
- In particular, data can be serialized to/from String, ByteString, file, memory-mapped file, memory buffer, another process

Further reading

- Documentation page: http://haskell.org/haskellwiki/Library/AltBinary
- Download: http://www.haskell.org/library/Streams.tar.gz http://www.haskell.org/library/StreamsBeta.tar.gz

Contact

⟨Bulat.Ziganshin@gmail.com⟩

4.7.3 Compression-2006 (was: Compression-2005)

Report by:	Bulat Ziganshin
Status:	stable

Features of the Compression-2006 Library:

- easy and uniform access to most competitive compression algorithms as of November'06: 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:

```
compressWithHeader
  "ppmd:10:48mb" (hGetBuf stdin) (hPutBuf stdout)
```

with decompressor program:

```
decompressWithHeader
  (hGetBuf stdin) (hPutBuf stdout)
```

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.

Compared to the previous version, I have upgraded the LZMA part of the library to use the LZMA 4.43 library that significantly improved the speed and compression ratio over old versions.

Further reading

- Documentation: http://haskell.org/haskellwiki/Library/Compression
- Download: http://www.haskell.org/library/CompressionLibrary.tar.gz

Contact

⟨Bulat.Ziganshin@gmail.com⟩

4.7.4 The Haskell Cryptographic Library

Report by:	Dominic Steinitz

The current release remains 3.0.3. However, there have been quite a few contributions and there is now a SHA-2 implementation as well as some performance improvements to SHA-1. There is now also a trac at http://hackage.haskell.org/trac/crypto.

As a result of the performance improvements, the interface to SHA-1 is now different from MD5 and the whole library needs a rethink. Unfortunately, I don't have the time to undertake any work on it at the moment and it is not clear when I will have time. I'm therefore looking for someone to take over the role of keeping the repository up-to-date with contributions, re-structuring the library and managing releases.

This release contains:

- DES
- Blowfish
- AES
- Cipher Block Chaining (CBC)
- PKCS#5 and nulls padding
- SHA-1
- MD5
- RSA
- OAEP-based encryption (Bellare-Rogaway)
- PKCS#1v1.5 signature scheme
- ASN.1
- PKCS#8
- X.509 Identity Certificates
- X.509 Attribute Certificates

Further reading

http://www.haskell.org/crypto

4.7.5 2LT: Two-Level Transformation

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

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

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

- A library of two-level transformation combinators. These combinators are used to compose transformation systems which, when applied to an input type, produce an output type, together with the conversion functions that mediate between input and out types.

- Front-ends for XML and SQL. These front-ends support (i) reading a schema, (ii) applying a two-level transformation system to produce a new schema, (iii) convert a document/database corresponding to the input schema to a document/database corresponding to the output schema, and *vice versa*. Referential constraints and primary key information are propagated through the schema transformation.

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

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

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

Efforts are underway to add further front-ends to 2LT, e.g. for XPath and VDM-SL, and to extend the SQL front-end.

Further reading

Project URL: http://wiki.di.uminho.pt/wiki/bin/view/PURe/2LT

- Alcino Cunha, José Nuno Oliveira, Joost Visser. *Type-safe Two-level Data Transformation*. Formal Methods 2006.

- Alcino Cunha, Joost Visser. Strongly Typed Rewriting For Coupled Software Transformation. RULE 2006.

- Pablo Berdaguer, Alcino Cunha, Hugo Pacheco, Joost Visser. *Coupled Schema Transformation and Data Conversion For XML and SQL*. PADL 2007.

- Alcino Cunha and Joost Visser. *Transformation of Structure-Shy Programs, Applied to XPath Queries and Strategic Functions*, Draft, 2006.

4.8 User interfaces

4.8.1 wxHaskell

Report by:	Jeremy O'Donoghue

A new team is adding support for the latest tools to wxHaskell, a mature and full-featured Haskell GUI binding.

Project members: Eric Kow, Mads Lindstroem, Shelarcy, Tim Docker, Frank Berthold.

wxHaskell is a stable and highly featured Haskell binding to the wxWidgets cross-platform GUI toolkit, originally developed by Daan Leijen and others.

The main benefits of using wxHaskell in a Haskell GUI project include:

- Many widgets have high-level Haskell bindings which bridge much of the impedence mismatch between Haskell code and typical (imperative) GUI code.

- Support for most of the (extensive) GUI functionality of wxWidgets

- Native look and feel on all supported platforms (Windows, OS X, Unix/Linux) due to the use of native widgets wherever possible.

- Straightforward to deploy, as only a small set of libraries needs to be distributed with the application (e.g. just two DLLs on Windows).

While it is (in our opinion) a superb piece of software, wxHaskell has recently suffered from a lack of maintenance. It did not compile against recent versions of GHC or wxWidgets and lacked Unicode support.

The new team is doing its best to rectify this. We have so far implemented: support for a couple of additional widgets; preliminary Unicode support; support

for recent versions of wxWidgets (up to 2.6.3) and very preliminary support for GHC 6.6.

With the help and support of Daan and Simon Marlow, we are now able to host wxHaskell development via Darcs patches at http://darcs.haskell.org/wxhaskell, and to administer the wxHaskell website and mailing lists (at Sourceforge).

The latest updates are as yet only available at darcs.haskell.org, although we plan occasional updates of Sourceforge CVS for those who prefer it, and will provide binaries when we are confident that we have achieved a good level of stability on all platforms.

Immediate plans are to Cabalize the build process, to improve Unicode support and to increase the number and complexity of sample programs.

Further reading

http://wxhaskell.sourceforge.net

4.8.2 Gtk2Hs

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

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

GUIs written using Gtk2Hs use themes to resemble the native look on Windows and, of course, various desktops on Linux, Solaris and FreeBSD. Gtk+ and Gtk2Hs also support MacOS X (it currently uses the X11 server but a native port is in progress).

Gtk2Hs features:

- automatic memory management (unlike some other C/C++ GUI libraries, Gtk+ provides proper support for garbage-collected languages)
- Unicode support
- anti-aliased drawing on screen, PDF, PS, etc. using Cairo
- extensive reference documentation
- an implementation of the Paul Hudak's Haskell School of Expressions graphics API
- support for the Glade visual GUI builder
- bindings to some Gnome extensions: GConf, a source code editor widget and a widget that embeds the Mozilla, Firefox and xulrunner rendering engines
- an easy-to-use installer for Windows
- packages for Fedora Core (→ 7.4.1), Gentoo (→ 7.4.3), Debian, FreeBSD and ArchLinux

The Gtk2Hs library is actively maintained and developed. We have completed a new API for the list and tree widgets. The data of these widgets is now stored in Haskell land as lists and as rose trees (`Data.Tree`) which makes manipulating the contents of the widgets much easier and, in contrast to the C interface, statically typed. We are currently working on providing a new interface for handling signals (callbacks from widgets). The old-style signaling mechanism will be deprecated in the next release as will many other functions that were superseeded by the attributes-based API. We anticipate that this will simplify the documentation drastically, thereby making Gtk2Hs easier to use.

The releases of Gtk2Hs are tested to run on Windows, Linux, MacOS X (PPC), FreeBSD, OpenBSD and Solaris. Due to the substantial additions, the next release of Gtk2Hs is slightly delayed, but should happen before Christmas. This release will break backwards compatability with applications using the old list and tree API. The API of Gtk2Hs should be stable thereafter, leading up to a version 1.0 release. We encourage people to use the darcs development version and to test the new list and tree API. Thanks goes to those who have already done so!

Further reading

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

4.8.3 hscurses

Report by:	Stefan Wehr
Status:	stable/beta

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

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

The binding itself is stable; however, the widget library is still beta. Volunteers are welcome to improve and extend the widget library. The build system now uses Cabal.

Further reading

http://www.informatik.uni-freiburg.de/~wehr/haskell/

4.9 (Multi-)Media

4.9.1 HOpenGL – A Haskell Binding for OpenGL and GLUT

Report by:	Sven Panne
Status:	stable, actively maintained

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

HOpenGL includes the simple GLUT UI, which is good to get you started and for some small to medium-sized projects, but HOpenGL doesn't rival the GUI task force efforts in any way. Smooth interoperation with GUIs like gtk+hs or wxHaskell (→ 4.8.1) on the other hand is a goal, see e.g. http://wxhaskell.sourceforge.net/samples.html#opengl

The feature highlights of HOpenGL are:

- Pure Haskell 98 + FFI, so it works on all Haskell platforms (GHC, Hugs, ...)
- No dependencies on external tools like GreenCard
- Almost complete OpenGL 2.1 support, including buffer objects and shaders
- A few dozen extensions
- A clean API, centered around OpenGL's notion of state variables
- Extensive hyperlinked online documentation
- Supports freeglut-only features, too

HOpenGL is available as two separate Cabal packages (OpenGL and GLUT) and is extensively tested on x86 Linux and Windows. The packages reportedly work on Solaris, FreeBSD, OpenBSD (→ 7.4.2), and Mac OS X, too.

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

Further reading

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

4.9.2 HOpenAL – A Haskell Binding for OpenAL and ALUT

Report by:	Sven Panne
Status:	stable, actively maintained

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

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

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

Further reading

http://www.openal.org/

4.9.3 Haskore revision

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

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

1. Basic Cabal support.
2. The `Music` data type has been generalized in the style of Hudak's "polymorphic temporal media."
3. The `Music` data type has been made abstract by providing functions that operate on it.
4. The notion of instruments is now very general. There are simple predefined instances of the `Music` data type, where instruments are identified by Strings or General MIDI instruments, but any other custom type is possible, including types with instrument specific parameters.
5. Support for CSound orchestra files has been improved and extended, thus allowing instrument design in a signal-processing manner using Haskell, including feedback and signal processors with multiple outputs.
6. Initial support for the real-time software synthesizer SuperCollider through the Haskell interface.

7. The AutoTrack project has been adapted and included.

8. Support for infinite `Music` objects is improved. CSound may be fed with infinite music data through a pipe, and an audio file player like Sox can be fed with an audio stream entirely rendered in Haskell. (See Audio Signal Processing project (→ 6.15).)

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

Future plans

- Split into a core package and add-ons, as soon as Cabal supports that.
- Generate note sheets, say via Lilypond.
- Allow modulation of instruments similar to the controllers in the MIDI system.
- Microtonal music.
- Connect to other Haskore related projects.

Further reading

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

4.10 Web and XML programming

4.10.1 HAppS – Haskell Application Server

Report by: S. Alexander Jacobson

HAppS is a framework for developing Internet services quickly, deploying them easily, scaling them massively, and managing them ziplessly. Web, persistence, mail, DNS and database servers are all built-in so you can focus on app development rather than integrating and babysitting lots of different servers/services (the Haskell type system keeps everything consistent).

- **HTTP Application Serving**

  Performs better than Apache/PHP in our informal benchmarks (thanks to Data.ByteString), handles large (video) files and lazy (javascript) streaming, supports HTTP-Auth, and more. It's part of your app so you don't need to deal with separate configuration and management of an HTTP server. Note: If you really need Apache on port 80 for some reason, it's easy to configure it to proxy to your HAppS app running on another port.

- **SMTP Sending (Relaying) with built in DNS resolver**

  Integrating outbound SMTP directly into your app means that you can stop worrying about configuration and uptime of separate mail and DNS servers. If your HAppS app is running, mail is being sent. If it is rebooted, nothing is lost. If mail is temporarily undeliverable, it does exponential backoff and tries again later.

- **SMTP Receiving (no more .procmail complexity)**

  Stop worrying about whether a separate mail server is up and stop dealing with .procmail or other user level inbound mail configuration hackery. HAppS can operate as an inbound SMTP server, converting inbound envelopes into just another event for your application to process. And, if you need a separate mail server on port 25, it should be much easier to configure it to SMTP relay mail to your HAppS app handling SMTP on a different port (you still avoid extra .procmail complexity/annoyance).

- **Apps as Simple State Transformers**

  HAppS keeps your application development very simple. You represent state with the Haskell data structure you find most natural for that purpose. Your app then is just a set of state transformer functions (in the MACID Monad) that take an event and state as input and that evaluate to a new state, a response, and a (possibly null) set of sideeffects.

- **XML/XSLT to Separate Application Logic and Presentation**

  HAppS lets you focus on application logic and lets you defer presentation entirely to XSLT, JSON, Flapjax, etc. HAppS converts automatically from inbound protocol level event types e.g. url-encoded HTTP requests to inbound application level event types e.g. ChangePassword. Similarly, it converts automatically from outbound application events like PasswordChanged and outbound protcol events like HTTP responses or SMTP messages. It even knows to apply XSLT server side for XML outbound SMTP messages and browsers that don't support XSLT client side. Currently, you still have to write instances for FromMessage and ToElement, but we hope to make that automatic soon.

- **ACID Persistence, Concurrency. At-least-Once side-effects.**

  With HAppS you need don't to spend time marshalling data into and out of external RDBMSs to get ACID semantics (concurrent-access) for your data. HAppS treats all events as atomic and puts them in a total order so you never need to worry about concurrency (isolation). HAppS achieves durability by state checkpointing and write-ahead logging events. End-users can never be confused by a server reboot because HAppS won't execute responses or side effects until their driving events have been logged. HAppS also tracks which side-effects have completed. If the server is rebooted before a side-effect completes, HAppS will retry on recovery. (This sophisticated side-effect functionality may be unique to HAppS)

- **(Experimental) Relational Table and Index in Haskell**

  Do relational operations (type) safely on in-memory Haskell Data.Set(s) rather than dealing with an external SQL relational database. Define custom indices for your Haskell datatypes (e.g. geographic/geometric types). Use in combination with MACID for a robust relational DBMS customized for your application.

- **Coming Soon: No need for server architecture (thanks to Amazon)**

  We are almost done with changes to the back end of HAppS so that apps will be able to run unchanged on Amazon's S3 (http://aws.amazon.com/s3) and EC2 (http://aws.amazon.com/ec2). The result will be massive scalability and superior reliability without you having to lift a finger or walk into a data center.

Example applications written on top of HAppS include a wiki that's included in the tutorial and pass.net (→ 4.10.2), an authentication webapp that improves upon the idea of confirmation emails.

HAppS version 0.8.4 was released on October 12th, 2006.

The October 12th release includes examples demonstrating new features such as user login, blocking IO, extended session support, and more.

The latest stable release can always be found on http://HAppS.org/.

The latest development version can be acquired with: `darcs get -partial` http://happs.org/HAppS

Further reading

- Website
  http://happs.org/
- Discussion Group
  http://groups.google.com/group/HAppS/
- pass.net
  http://pass.net

4.10.2 Pass.Net

Report by:	S. Alexander Jacobson

Pass.Net provides web sites with a simple shared web API to manage user logins, confirmation emails, forgotten passwords, etc. Most application frameworks don't have complete libraries to cover all of this functionality.

Outsourcing this to Pass.net means less complexity in your application and less worrying about mail delivery, mail server integration, etc.

It also means your users don't need to confirm their email for *yet another* website if they've confirmed their email address on any other site that uses Pass.Net.

Pass.Net is currently beta. We expect it to be fully live and reliable by the end of the year. Pass.Net is written in Haskell using HAppS (→ 4.10.1) and provides an easy to use Haskell library for HAppS user. Clients in python, php, and java coming soon.

The source code for all of Pass.net is available at http://pass.net/s/repo.

4.10.3 Converter of Yhc Core to Javascript (ycr2js)

Report by:	Dimitry Golubovsky
Status:	experimental

Converter of Yhc Core to Javascript (further referred to as ycr2js) is a sub-project of the York Haskell Compiler (further referred to as Yhc) Project. It is aimed to create a tool to convert an arbitrary Haskell program into Javascript which in turn may be executed in a Web browser.

Conversion from Haskell to Javascript is achieved in two steps: a Haskell source is translated into Yhc Core by Yhc, and then the Core is translated to Javascript by ycr2js. Additional tools are provided to embed generated Javascript onto a Web page.

This allows to develop Internet applications entirely in Haskell (solutions for the server side have been around for a while, such as HAppS (→ 4.10.1) and HWS). A close analog to ycr2js is "HSP Client Side", which provides a domain-specific language (named HJScript) to define Javascript constructs to be executed at the client side, but not the ability to execute arbitrary Haskell code in a Web browser.

Further reading

- Yhc Core:
  http://haskell.org/haskellwiki/Yhc/API/Core
- The Wiki page http://haskell.org/haskellwiki/Yhc/Javascript contains some examples of Haskell programs translated into Javascript.

4.10.4 HaXml

Report by:	Malcolm Wallace
Status:	stable, maintained

HaXml provides many facilities for using XML from Haskell. The public stable release is 1.13.2, with support for building via Cabal, and for ghc-6.6.

In the unstable development version (currently at 1.17, also available through a darcs repository) we have been experimenting successfully with improvements to the secondary parsing stage, where the generic XML tree is re-parsed into typed Haskell trees. We now get good error messages if the parse fails, and the tools DtdToHaskell and DrIFT (→ 3.5) have been updated to

use the new framework. Lazy variations of the basic parser and pretty-printer also now exist, which *much* reduces the memory needed if you are processing documents in a linear (top-to-bottom) fashion – of course this also means it is now possible to deal with really large inputs.

Some minor work still remains to tidy things up before the development version is tagged as stable, but no further major changes are planned before that.

Further reading

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

4.10.5 Haskell XML Toolbox

Report by:	Uwe Schmidt
Status:	fifth major release (current release: 6.1)

Description

The Haskell XML Toolbox is a collection of tools for processing XML with Haskell. It is itself purely written in Haskell 98. The core component of the Haskell XML Toolbox is a validating XML-Parser that supports almost fully the Extensible Markup Language (XML) 1.0 (Second Edition), There is validator based on DTDs and a new more powerful validator for Relax NG schemas.

The Haskell XML Toolbox bases on the ideas of HaXml (→ 4.10.4) and HXML, but introduces a more general approach for processing XML with Haskell. Since release 5.1 there is a new arrow interface similar to the approach taken by HXML. This interface is more flexible than the old filter approach. It is also safer, type checking of combinators becomes possible with the arrow interface.

Features

- Validating XML parser
- Very liberal HTML parser
- XPath support
- Full Unicode support
- Support for XML namespaces
- Flexible arrow interface with type classes for XML filter
- Package support for ghc
- Native Haskell support of HTTP 1.1 and FILE protocol
- HTTP and access via other protocols via external program curl
- Tested with W3C XML validation suite
- Example programs for filter and arrow interface
- Relax NG schema validator based on the arrows interface
- A HXT Cookbook for using the toolbox and the arrow interface
- Basic XSLT support (next release)

Current Work

A master thesis has been finished developing an XSLT system. The reuslt is a rather complete implementation of an XSLT transformer system. Only minor features are missing. The implementaion consists of about only 2000 lines of Haskell code. The XSLT module will be included in the next HXT release.

A second master student's project will be finished until end of 2006. The title is *A Dynamic Webserver with Servlet Functionality in Haskell Representing all Internal Data by Means of XML*. HXT with the arrows interface has been used for all internal data processing. The results of this work will be available with the next HXT release.

The next HXT release is planned for December 2006.

Further reading

The Haskell XML Toolbox Web page (http://www.fh-wedel.de/~si/HXmlToolbox/index.html) includes downloads, online API documentation, a cookbook with nontrivial examples of XML processing using arrows and RDF documents, and master thesises describing the design of the toolbox, the DTD validator and the arrow based Relax NG validator. A getting started tutorial about HXT is avaliable in the Haskell Wiki (http://www.haskell.org/haskellwiki/HXT).

4.10.6 WASH/CGI – Web Authoring System for Haskell

Report by:	Peter Thiemann

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

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

- documented preprocessor for translating markup in syntax close to XHTML syntax into WASH/HTML

Completed Items are:
- fully cabalized
- WASH server pages with a modified version of Simon Marlow's `hws` web server; the current prototype supports dynamic compilation and loading of WASH source (via Don Stewart's hs-plugins (→ 4.4.1)) as well as the implementation of a session as a continually running server thread
- Transactional interface to server-side variables and to databases. The interface is inspired by the work on STM (software transactional memory), but modified to be useful in the context of web applications. The interface relies on John Goerzens hdbc package and its PostgreSQL driver.

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

Further reading

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

4.10.7 HAIFA

Report by: Simon Foster

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

HAIFA has been undergoing some substantial work since the last HCAR. Support for extensible hooks has now been dropped, as this makes writing and invoking serializers much simpler and its usefulness was questionable. Extensible hook support was designed primarily as a method of encoding meta-data into a serialization tree using type-classes unknown when the base serializers were written to bring in meta-data. Due to Haskell's static type-system it is unlikely this could ever have been put to use, and most of its functionality can probably be achieved with value-level generics via meta-data tables.

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

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

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

Further reading

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

5 Tools

5.1 Foreign Function Interfacing

5.1.1 FFI Imports Packaging Utility

Report by:	Dimitry Golubovsky
Status:	pre-release

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

The utility is a recent addition to the HSFFIG package.

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

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

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

Current Status

Pre-release. The utility is available from darcs repo (→ 6.4) only. The package installs as HSFFIG-1.1. Updated HSFFIG is also available from this package.

Further reading

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

5.1.2 C→Haskell

Report by:	Manuel Chakravarty
Status:	active

C→Haskell is an interface generator that simplifies the development of Haskell bindings to C libraries. It reads C header files to automate many tedious aspects of interface generation and to minimise the opportunity for introducing errors when translating C declarations to Haskell.

The darcs repository (→ 6.4) of C→Haskell is now at http://darcs.haskell.org/c2hs with Duncan Coutts being a second maintainer. The last few months didn't see much C→Haskell development due to work on other projects; however, we have plans for the future. More information is at http://www.cse.unsw.edu.au/~chak/haskell/c2hs/.

5.2 Scanning, Parsing, Analysis

5.2.1 Frown

Report by:	Ralf Hinze
Status:	beta, maintained

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

Its salient features are:

- The generated parsers are time and space efficient. On the downside, the parsers are quite large.
- Frown generates four different types of parsers. As a common characteristic, the parsers are *genuinely functional* (i.e. 'table-free'); the states of the underlying LR automaton are encoded as mutually recursive functions. Three output formats use a typed stack representation, one format due to Ross Paterson (`code=stackless`) works even without a stack.
- Encoding states as functions means that each state can be treated individually as opposed to a table driven-approach, which necessitates a uniform treatment of states. For instance, look-ahead is only used when necessary to resolve conflicts.
- Frown comes with debugging and tracing facilities; the standard output format due to Doaitse Swierstra (`code=standard`) may be useful for teaching LR parsing.

- Common grammatical patterns such as repetition of symbols can be captured using *rule schemata*. There are several predefined rule schemata.

- Terminal symbols are arbitrary variable-free Haskell patterns or guards. Both terminal and nonterminal symbols may have an arbitrary number of synthesized attributes.

- Frown comes with extensive documentation; several example grammars are included.

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

The current release is version 0.6.1.

Further reading

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

5.2.2 Alex version 2

Report by:	Simon Marlow
Status:	stable, maintained

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

The latest release is version 2.0.1. Alex version 2.1 is currently in release-candidate mode, and is expected to be released shortly.

Recent changes:

- Alex is now in a Darcs repository (→ 6.4), here: http://cvs.haskell.org/darcs/alex.

- Happy has a new build system, based on Cabal. If you have GHC 6.4.2 or later (or Cabal 1.1.4 or later), then you should be able to build and install Alex on any platform. On Windows, Perl is required in addition to GHC for building, but that is all.

- There was a slight change in the error semantics, to enable more informative error messages.

Further reading

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

5.2.3 Happy

Report by:	Simon Marlow
Status:	stable, maintained

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

The latest release is 1.15, released 14 January 2005. Version 1.16 is currently in release-candidate mode, and is expected to be released very shortly. Version 1.16 is required to build GHC.

Since that release, the following changes have happened:

- Happy is now in a Darcs repository (→ 6.4), here: http://darcs.haskell.org/happy.

- Happy has a new build system, based on Cabal. If you have GHC 6.4.2 or later (or Cabal 1.1.4 or later), then you should be able to build and install Alex on any platform. On Windows, Perl is required in addition to GHC for building, but that is all.

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

- Attribute Grammar support, contributed by Robert Dockins, has been added.

Further reading

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

5.2.4 SdfMetz

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

SdfMetz supports grammar engineering by calculating grammar metrics and other analyses. Currently it supports two different grammar formalisms (SDF and DMS) from which it calculates size, complexity, structural, and ambiguity metrics. Output is a textual report or in Comma Separated Value format. The additional analyses implemented are visualization, showing the non-singleton levels of the grammar, or printing the grammar graph in DOT format. The definition of all except the ambiguity and the NPath metrics were taken from the paper *A metrics suite for grammar*

based-software by James F. Power and Brian A. Malloy. The ambiguity metrics were defined by the tool author exploiting specific aspects of SDF grammars and the NPath metric definition was taken from the paper *NPATH: a measure of execution path complexity and its applications*.

A web-based interface is planned and more metrics will be added. A front-end to other grammar formalism (yacc and antlr) is also planed, being the yacc front-end currently under development. As longer term project, it is expected to fuse the *SdfMetz* and *XsdMetz* in a single tool.

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

Further reading

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

5.2.5 XsdMetz: metrics for XML Schema

Report by:	Joost Visser
Status:	maintained

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

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

Further reading

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

5.3 Transformations

5.3.1 Term Rewriting Tools written in Haskell

Report by:	Salvador Lucas

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

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

Status

The previous work lead to the following tools:

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

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

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

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

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

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

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

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

All these tools have been written in Haskell (mainly developed using Hugs and GHC) and use popular Haskell libraries like hxml-0.2, Parsec, RegexpLib98, wxHaskell (→ 4.8.1).

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

- Building .NET GUIs for Haskell applications. B. Alarcón and S. Lucas. 6th International Conference on .NET Technologies, to appear, 2006.
- Abstract Diagnosis of Functional Programs M. Alpuente, M. Comini, S. Escobar, M. Falaschi, and S. Lucas Selected papers of the International Workshop on Logic Based Program Development and Transformation, LOPSTR'02, LNCS 2664:1-16, Springer-Verlag, Berlin, 2003.
- OnDemandOBJ: A Laboratory for Strategy Annotations M. Alpuente, S. Escobar, and S. Lucas 4th International Workshop on Rule-based Programming, RULE'03, Electronic Notes in Theoretical Computer Science, volume 86.2, Elsevier, 2003.
- Connecting remote termination tools M. Alpuente and S. Lucas 7th International Workshop on Termination, WST'04, pages 6–9, Technical Report AIB-2004-07, RWTH Aachen, 2004.
- MU-TERM: A Tool for Proving Termination of Context-Sensitive Rewriting S. Lucas 15th International Conference on Rewriting Techniques and Applications, RTA'04, LNCS 3091:200-209, Springer-Verlag, Berlin, 2004.
- A Rule-based System for Web site Verification. Demis Ballis and Javier García-Vivó. 1st International Workshop on Automated Specification and Verification of Web Sites, WWV'05, Valencia (SPAIN). Electronic Notes in Theoretical Computer Science, to appear, 2005.

5.3.2 HaRe – The Haskell Refactorer

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

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

Our project, *Refactoring Functional Programs* has as its major goal to build a tool to support refactorings in Haskell. The HaRe tool is now in its third major release. HaRe supports full Haskell 98, and is integrated with Emacs (and XEmacs) and Vim. All the refactorings that HaRe supports, including renaming, scope change, generalisation and a number of others, are *module aware*, so that a change will be reflected in all the modules in a project, rather than just in the module where the change is initiated. The system also contains a set of data-oriented refactorings which together transform a concrete `data` type and associated uses of pattern matching into an abstract type and calls to assorted functions. The latest snapshots support the hierarchical modules extension, but only small parts of the hierarchical libraries, unfortunately. The version about to be released (at the time of writing) works with GHC 6.4.2.

In order to allow users to extend HaRe themselves, the latest releases of HaRe include an API for users to define their own program transformations, together with Haddock (→ 5.5.6) documentation. Please let us know if you are using the API.

There have been some recent developments for adding program slicing techniques to HaRe. These techniques include a refactoring to split functions returning tuples into separate definitions, and to also put them back together again. There have also been some new refactorings added which work on data types: adding a constructor to a data type and converting a data type into a newtype. The immediate aim for the development of HaRe is to support a number of type-based refactorings.

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

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

Further reading

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

5.3.3 VooDooM

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

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

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

Re-engineering is supported through the application of transformation rules to the datatypes to ob-

tain an equivalent relational representation. The relational representation can be exported as VDM-SL datatypes (inserted back into th e original specification) and/or SQL table definitions (can be fed to a relational DBMS).

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

Currently, a reimplementation of the re-engineering functionality of VooDooM is being undertaken, based on so-called two-level transformations, as supported by the 2LT project (→ 4.7.5).

As future work the implementation is expected of both XML and Haskell generation.

Further reading

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

5.4 Testing and Debugging

5.4.1 Haskell Program Coverage

Report by:	Andy Gill
Status:	released, maintained, in active development

Over the summer Colin Runciman visited Galois (→ 7.1.2) for a sabattical. Andy Gill and Colin teamed up to design and implement a Haskell coverage tool for use by Galois to support QA efforts. The result was Haskell Program Coverage (Hpc).

Hpc is a tool-kit to record and display Haskell Program Coverage. Hpc includes tools that instrument Haskell programs to record program coverage, run instrumented programs, and display the coverage information obtained.

Hpc provides coverage information of two kinds: source coverage and boolean-control coverage. Source coverage is the extent to which every part of the program was used, measured at three different levels: declarations (both top-level and local), alternatives (among several equations or case branches) and expressions (at every level). Boolean coverage is the extent to which each of the values True and False is obtained in every syntactic boolean context (ie. guard, condition, qualifier).

Hpc displays both kinds of information in two different ways: textual reports with summary statistics (hpc-report) and sources with colour mark-up (hpc-markup). For boolean coverage, there are four possible outcomes for each guard, condition or qualifier: both True and False values occur; only True; only False; never evaluated. In hpc-markup output, highlighting with a yellow background indicates a part of the program that was never evaluated; a green background indicates an always-True expression and a red background indicates an always-False one.

Hpc provides a Haskell-to-Haskell translator as a means for building instrumented binaries for gathering coverage information, and an Hpc option already checked into GHC 6.7 will make gathering coverage over GHC specific Haskell code possible in the near future.

The file formats use by Hpc are simple and well documented. The intent is that other tools can be quickly built that process coverage information in creative ways.

GHC has been sucessfully bootstrapping using Hpc, and Hpc has already be deployed internally in Galois in a number of places. In the future expect to see tighter integration between Haskell testing tools and Hpc as obtaining coverage results for test runs becomes standard practice in Haskell development.

Further reading

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

5.4.2 Hat

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

The Haskell tracing system Hat is based on the idea that a specially compiled Haskell program generates a trace file alongside its computation. This trace can be viewed in various ways with several tools: hat-observe, hat-trail, hat-detect, hat-delta, hat-explore, hat-cover, hat-anim, black-hat, hat-nonterm ... Some views are similar to classical debuggers for imperative languages, some are specific to lazy functional language features or particular types of bugs. All tools inter-operate and use a similar command syntax.

Hat can be used both with nhc98 (→ 2.3) and ghc (→ 2.1). Hat was built for tracing Haskell 98 programs, but it also supports some language extensions (FFI, MPTC, fundeps, hierarchical libs). A tutorial explains

how to generate traces, how to explore them, and how they help to debug Haskell programs.

Bugfixes and new viewing tools are being added continuously. Now parts of Hat can be used under Windows. Hat-GUI repackages a number of existing viewing tools under a graphical user interface. Hat-delta extends algorithmic debugging with heuristics, tree compression and a new representation of functional values as finite maps.

Development of Hat recently moved to using darcs (→ 6.4). In October 2006 an interim release of 2.05 was published, specifically to allow Hat to compile with ghc-6.6 (→ 2.1), but we hope to put out a new full release 2.06 in the next few months, which will be much more compatible with the current state of the common library packages.

Further reading

- Hat Day 2006 http://www.cs.kent.ac.uk/people/staff/oc/TraceTheory/hatDay2006.html
- Thomas Davie and Olaf Chitil: Display of Functional Values for Debugging. Draft Proceedings of the 18th International Workshop on Implementation and Application of Functional Languages, IFL 06.
- Thomas Davie and Olaf Chitil: One Right Does Make a Wrong. Trends in Functional Programming, TFP '06. http://www.cs.nott.ac.uk/~nhn/TFP2006/Papers/18-DavieChitil-OneRightDoesMakeAWrong.pdf
- A Theory of Tracing Pure Functional Programs http://www.cs.kent.ac.uk/~oc/traceTheory.html
- http://www.haskell.org/hat
- darcs get http://darcs.haskell.org/hat

5.4.3 buddha

Report by:	Bernie Pope
Status:	inactive

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

Buddha is freely available as source and is licensed under the GPL. There is also a Debian package, as well as ports to Free-BSD, Darwin and Gentoo (→ 7.4.3).

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

Further reading

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

5.4.4 SmallCheck: another lightweight testing library in Haskell

Report by:	Colin Runciman

SmallCheck is similar to QuickCheck (Claessen and Hughes 2000–) but instead of testing for a sample of randomly generated values, SmallCheck tests properties for all the finitely many values up to some depth, progressively increasing the depth used. As well as guaranteeing minimal counter-examples, the different approach to test-data generation makes it easier to define generators for user-defined types, allows the use of existential quantifiers and enables more information to be displayed about functional values.

The SmallCheck prototype was written in Summer 2006 during a visit to Galois Connections (→ 7.1.2). Feedback from users has prompted improvements, and the most recent version is 0.2 (November 2006). Compared with version 0.1 there is a wider choice of test-drivers and more pre-defined test-data generators. SmallCheck 0.2 is freely available, with illustrative examples, from http://www.cs.york.ac.uk/fp/smallcheck0.2.tar.

5.4.5 Dr Haskell

Report by:	Neil Mitchell

Dr Haskell is a tool to detect common mistakes in beginner Haskell programs. For example, beginners might not be aware of `concatMap`, and instead use `concat (map f x)`. Dr Haskell does not make any changes to the source code, it merely suggests places where things could be improved. A database is included which knows about 20 suggestions, but more can be added easily, with a simple syntax.

Further reading

http://www-users.cs.york.ac.uk/~ndm/projects/drhaskell.php

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

ghc (→ 2.1), hbc, and nhc98 (→ 2.3), allowing multiple installed versions of compilers to be easily selected from.

A recent public version: 3.13, contains bugfixes for building with ghc-6.6. Maintenance continues at darcs.haskell.org.

Further reading

http://haskell.org/hmake/

5.5.2 Ruler

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

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

- To keep the implementation and LaTeX rendering of type rules consistent.
- To allow an incremental specification (necessary for the stepwise description employed by EHC).

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

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

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

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

Current activities

Arie Middelkoop continues with the development of the Ruler system as part of his Microsoft Research Scholarship PhD grant. He will investigate the specification of type rules in a partitioned (stepwise an aspectwise) fashion, and the incorporation of solving strategies for typing rules.

Further reading

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

5.5.3 cpphs

Report by:	Malcolm Wallace
Status:	stable, maintained

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

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

Current release is 1.3, containing some minor bugfixes and a new `-cpp` option to accept original cpp flag syntax, so you can use it as a truly drop-in replacement.

Further reading

http://haskell.org/cpphs

5.5.4 Visual Haskell

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

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

The first release of Visual Haskell, version 0.0, was announced on 20 September 2005. It can be obtained from the main Visual Haskell page, here: http://www.haskell.org/visualhaskell/. In order to use Visual

Haskell, you need an x86 machine running Windows, and Visual Studio .NET 2003.

Following a relaxation in the license under which Microsoft's Visual Studio SDK is released, we are now able to distribute the source to the plugin under a BSD-style license. The sources are in a darcs (→ 6.4) repository here: http://darcs.haskell.org/vshaskell/. Why not take a look and see what lengths you have to go to in order to write Haskell code that plugs into Visual Studio!

Now a there is a prerelease version that is available for both Visual Studio 2003 and Visual Studio 2005. It will be distributed with GHC-6.6. There is a new installer with updated (BSD) license. The installer is now bundled with both the normal and the profiling libraries.

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

5.5.5 Haskell support for the Eclipse IDE

Report by:	Leif Frenzel
Status:	working, though alpha

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

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

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

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

Further reading

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

5.5.6 Haddock

Report by:	Simon Marlow
Status:	stable, maintained

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

The latest release is verison 0.8, released October 10 2006.

Recent changes:

- Haddock is now in a Darcs repository (→ 6.4), here: http://darcs.haskell.org/haddock.
- Happy has a new build system, based on Cabal. If you have GHC 6.4.2 or later (or Cabal 1.1.4 or later), then you should be able to build and install Alex on any platform. On Windows, Perl is required in addition to GHC for building, but that is all.
- New in version 0.8: linking to source code from documentation, linking to wiki from documentation, generating output for Hoogle, and the `<<url>>` markup for including images.

Further reading

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

5.5.7 Hoogle – Haskell API Search

Report by:	Neil Mitchell
Status:	v3.0

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

Hoogle is still under active development, since the last HCAR substantial progress has been made towards version 4 – speeding up searches and offering many features requested by the users. Hoogle function database generation has also been integrated into Cabal, see the `haddock` command with the `-hoogle` flag.

Hoogle is available as a web interface, a command line tool and a lambdabot (→ 6.7) plugin.

Further reading

http://haskell.org/hoogle

5.5.8 SearchPath

Report by: S. Alexander Jacobson

Searchpath gives you automatic import chasing across the Internet for Haskell modules. Think of it as an internet wide version of the `-i` command line option for GHC. Rather than just specifying local file paths, you can specify locations out on the Internet for your compiler to find your modules. You don't need to worry about manually installing package after package, you only need a list of locations of packages (or parts of packages) you want to use, and let searchpath take care of the rest.

Detailed tutorial and more at http://www.haskell.org/haskellwiki/SearchPath. Also see the website at http://www.searchpath.org/.

6 Applications

6.1 FreeArc

Report by:	Bulat Ziganshin
Status:	beta

FreeArc is an archiver (like Info-Zip) written in Haskell that uses C compression libraries via the interface provided by the Compression-2006 library (→ 4.7.3).

At this moment, it's the best practical compressor in the world, several times faster and reaching better compression than WinRAR, 7-zip, WinRK, UHARC and any other program I know. Aside this, FreeArc provides a lot of features, including solid archives with fast updates, tunable compression level/algorithms, automatic selection of compression algorithm depending on file type, tunable sorting and grouping of files, SFX module and FAR MultiArc sub-plugin.

FreeArc sources have a lot of comments ... in Russian. If you know this language, these sources are an invaluable place for learning Haskell. Moreover, the program includes several modules that you may reuse in your program on BSD3 license:

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

Further reading

- Download:
  http://www.haskell.org/bz

Contact

⟨Bulat.Ziganshin@gmail.com⟩

6.2 h4sh

Report by:	Don Stewart
Status:	maintained

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

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

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

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

Further reading

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

6.3 Pugs

Report by:	Audrey Tang
Status:	active development

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

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

The Pugs team has over 200 committers from Haskell, Perl, Python, Ruby, JavaScript and other language communities; the *Learning Haskell* and *Introduction to Pugs* set of talks, published at the Pugs homepage, were also welcomed in several Open Source

conferences. Join us on irc.freenode.net `#perl6` to participate in the development!

Further reading

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

6.4 Darcs

Report by:	Eric Kow
Participants:	David Roundy
Status:	active development

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

David Roundy and a handful of interested users have been working on a new version of the theory of patches with better defined and more well-behaved conflict resolution. The two most actively pursued approaches so far are conflictors and tree-based conflicts. Meanwhile, the darcs community in general are working on day-to-day issues such as an improved interactions with external programs. A new release (1.0.9) will be coming out shortly, with GHC 6.6 support, several bug fixes and user interface improvements. Patches great and small would be heartily welcome!

Darcs is free software licensed under the GNU GPL.

Further reading

http://darcs.net

6.5 downNova

Report by:	Lemmih

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

Further reading

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

6.6 Hircules, an irc client

Report by:	Jens Petersen

Hircules is a gtk2-based IRC client built on gtk2hs (→ 4.8.2) and code from lambdabot (→ 6.7). The last release was version 0.3. I recently updated my tree to build with the current releases of ghc and gtk2hs and I am planning to import it to darcs.haskell.org soon to make it easier for other people to contribute patches.

Further reading

http://haskell.org/hircules/

6.7 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, unlambda interpreter, pointfree programming, dictd client, fortune cookies, Google search, online help and more.

New features since the last release include: redo/undo refactoring for monadic code, SmallCheck and QuickCheck support, a BF interpreter, persistent bindings via `@let`, free theorems generators, tiny-url support, many stability improvements.

Lambdabot is also able now to be embedded in GHCi, and runs online via an AJAX web interface.

Further reading

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

6.8 λFeed

Report by:	Manuel Chakravarty
Status:	active

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

6.9 yi

Report by:	Don Stewart
Status:	maintained

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

Further reading

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

6.10 Dazzle

Report by:	Martijn Schrage and Arjan van IJzendoorn

Dazzle is a graphical toolbox for Bayesian networks that is developed by the Decision Support System group of Utrecht University. It is written in Haskell and uses wxHaskell (→ 4.8.1) 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 Windows, Linux and Mac OS X. The project has produced several spin-offs: a progress indicator for pure algorithms, an abstraction for persistent documents, and the XTC library for typed controls. The Dazzle toolbox itself is closed source, but the spin-off libraries are available from the web page.

Further reading

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

6.11 INblobs – Interaction Nets interpreter

Report by:	Miguel Vilaca
Participants:	Miguel Vilaca and Daniel Mendes
Status:	active, maintained
Portability:	Windows, Linux and Mac OS X (depends on wxHaskell(→ 4.8.1))

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

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

The tool is being developed using the repository system Darcs (→ 6.4).

New features

- Mac OS X portability
- new reduction strategy
- templates for explicit memory management
- some bug fixes

Further reading

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

6.12 DoCon, the Algebraic Domain Constructor

Report by:	Serge Mechveliani

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

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

Real DoCon development has stopped before 2002. At the moment, only the GHC system-dependent changes are considered. Probably, there will be a new release (version 2.09) which runs under GHC (→ 2.1) 6.6 or later. This is a matter of correcting the GHC system usage in the manual.

Further reading

http://haskell.org/docon/

6.13 Dumatel, a prover based on equational reasoning

Report by:	Serge Mechveliani

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

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

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

Further reading

http://haskell.org/dumatel/

6.14 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.

The current release is version 1.11. Compiling with GHC 6.6 is possible, but requires a small change to the build system. Mainly for this reason, a new release 1.12 is planned, which probably will also include support for Cabal. Development continues slowly in the Subversion repository.

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

Further reading

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

6.15 Audio signal processing

Report by:	Henning Thielemann
Status:	experimental, active development

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

- a simple signal synthesis backend for Haskore (→ 4.9.3),
- experimental structures for filter networks,
- basic audio signal processing including some hard-coded frequency filters,
- advanced framework for signal processing supported by physical units, that is, the plain data can be stored in a very simple number format, even fixed point numbers, but the sampling parameters rate and amplitude can be complex types, like numbers with physical units,
- framework for inference of sample rate and amplitude, that is, sampling rate and amplitude can be omitted in most parts of a signal processing expression, they are inferred automatically, just as types are inferred in Haskell's type system. Although the inference of signal parameters needs some preprocessing, the framework preserves the functional style of programming. This approach is based on an explicitly maintained dictionary of signal parameters, which must be computed completely before any signal processing takes place. This forces all signal parameters to share the same type and prohibits infinitely many signal processors to be involved.

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

Future plans

- We try hard to get rid of the explicit dictionary in the sample parameter inference framework. We have some success on solving this problem, but sharing of signal data between signal processes is still the major problem.

- Design a common API to the Haskell synthesizer code, CSound support included in Haskore (→ 4.9.3), and the SuperCollider interface.
- Connect with the HaskellDSP library http://haskelldsp.sourceforge.net/.
- Hope on faster code generated by Haskell compilers. :-)

Further reading

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

6.16 hmp3

Report by:	Don Stewart
Status:	stable, maintained

hmp3 is a curses-based mp3 player frontend to mpg321 and mpg123. It is written in Haskell. It is designed to be simple, fast and robust. It's very stable.

hmp3 will now take advantage, transparently, of multiple cores, to run its separate threads, if compiled with the GHC 6.6 SMP runtime system.

Further reading

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

6.17 Testing Handel-C Semantics Using QuickCheck

Report by:	Andrew Butterfield
Participants:	Andrew Butterfield, Brian Corcoran
Status:	ongoing

The Handel-C Semantics Tool is a Haskell application that allows experimentation with formal semantic models of the hardware compiler language Handel-C, marketed by Celoxica Ltd. It has been used to evaluate to differing degrees three models: operational, denotational and hardware-oriented. It uses QuickCheck as a means for testing various key properties such as equivalence of the operational and denotational semantics, and the validity of certain algebraic laws for the language.

It is not yet publicly available – it is unsure what general interest there would be in this tool

We plan to revisit some of the tests, and then do the formal proofs!

Further reading

https://www.cs.tcd.ie/research_groups/fmg/moin.cgi/Handel_2dC_20Semantics_20Page

6.18 View selection for image-based rendering

Report by:	Yann Morvan
Status:	Part of a submitted Ph.D.

The initiative was part of a computer graphics research project aimed at proposing a perceptual view selection method for image-based rendering. Our approach was limited to applying functional programming to develop a complex graphics application, including a state of the art image-based renderer. We used Haskell with GHC and its OpenGL (→ 4.9.1) binding, adding a few wrappers for graphics hardware programming. Development proved agreeable, with almost no need for debugging. We had hoped to leverage lazy evaluation within the implementation of the view selection algorithm, but this didn't materialize. The project has been completed and there are presently no plans to dig further into the functional programming aspect of it, but it is a possibility. Source code is available on demand, as well as the Ph.D. manuscript, though it focuses very little on the functional aspect.

Further reading

https://www.cs.tcd.ie/~morvany/

7 Users

7.1 Commercial users

7.1.1 Bluespec tools for design of complex chips

Report by:	Rishiyur Nikhil
Status:	Commercial product

Bluespec, Inc. provides tools for chip design (ASICs and FPGAs) inspired by Haskell and Term Rewriting Systems. Bluespec also uses Haskell to implement many of its tools (over 85K lines of Haskell). Bluespec's products include synthesis, simulation and other tools for two languages:

- Bluespec SystemVerilog (BSV)
- ESE (ESL Synthesis Extensions to SystemC)

Both languages are based on a common semantic model: hardware behavior is expressed using *Rewrite Rules*, and inter-module communication is expressed using *Rule-based Interface Methods* (which allow rules to be composed from fragments that span module boundaries). Because rules are atomic, they eliminate a majority of the "timing errors" and "race conditions" that plague current hardware design using existing RTL languages like Verilog or VHDL. Rules also enable powerful reasoning about the functional correctness of systems. In other words, the concurrency model provided by rules is much more powerful and abstract than the low-level concurrency models provided by Verilog, VHDL and SystemC.

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

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

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

Status Bluespec SystemVerilog and its tools have been available since 2004, and Bluespec ESE since 2006. The tools are now in use by several major semiconductor companies (see Bluespec website or contact Bluespec for details) and several universities (including MIT, CMU, UT Austin, Virginia Tech, Indian Institute of Science, and U.Tokyo).

Availability Bluespec SystemVerilog and ESE tools are commercial tools sold by Bluespec, Inc. A free version of ESE, the SystemC-based product, that supports basic TRS rule simulation (i.e., without clock-scheduling, and without synthesis), is available with registration from the company website, complete with documentation, examples and training material. Bluespec, Inc. also makes all its tools easily available to academic institutions for teaching and research.

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

Further reading

- Company website:
  http://www.bluespec.com
- Publications:
  http://www.bluespec.com/technology/research.htm
  Bringing Declarative Programming into a Commercial Tool for Developing Integrated Circuits, Rishiyur Nikhil, Commercial Users of Functional Programming (CUFP), September 2006, slides of presentation at http://www.galois.com/cufp/
  MIT courseware, "Complex Digital Systems":
  http://www.csg.lcs.mit.edu/6.375 and
  http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-884Spring-2005/CourseHome/index.htm
  CMU courseware, "Hardware Systems Engineering":
  http://www.ece.cmu.edu/~ece744

7.1.2 Galois Connections, Inc.

Report by: Andy Adams-Moran

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

Galois develops software under contract, and every project (bar two) that we have ever done has used Haskell; the two exceptions used SML-NJ and OCaml, respectively. We've delivered tools, written in Haskell, to clients in industry and the U.S. government that are being used heavily. Some diverse examples: Cryptol, a domain-specific language for cryptography (with an interpreter and a compiler, with multiple targets); a GUI debugger for a specialized microprocessor; a specialized, high assurance web server, file store, and wiki for use in secure environments, and numerous smaller research projects that focus on taking cutting-edge ideas from the programming language and formal methods community and applying them to real world problems.

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

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

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

The good news is that our business is working quite well. As of Fall 2006, Galois is 21 engineers strong, with a support staff of 10. We've been profitable and experienced solid growth each of the last three years.

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

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

Further reading

http://www.galois.com/.

7.1.3 Aetion Technologies LLC

Report by: J. Garrett Morris

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

Our business model requires that we be able to rapidly prototype new systems as well as develop generic software foundations that we can extend to new markets as they open. We have found that Haskell fits both of these purposes; the majority of our codebase is written in Haskell and compiled using GHC.

Further reading

http://www.aetion.com/

7.1.4 Linspire

Report by: Clifford Beshers

The OS team at Linspire, Inc. uses Haskell as our preferred language for system tools. We have used O'Caml extensively as well, but are steadily migrating this code to Haskell.

Our largest project to date is our Debian package builder (aka autobuilder) in Haskell. The autobuilder is responsible for compiling all packages, which entails fetching source code from multiple source code control systems, building and caching clean chroot environments with the correct build dependencies, sorting the target package by build dependency to ensure they are built in the correct order, and so forth.

We are extending this system to be a package/OS release management system, where changes to packages can be grouped into sets and applied to an existing distribution (set of source and binary packages). The autobuilder is responsible for ensuring that all packages are rebuilt correctly for any source level change.

Other tools such as installer CD (ISO) builders, package dependency checkers are in progress. The goal is to make a really tight simple set of tools that will let developers contribute to Freespire, based on Debian tools whenever possible. Our hardware detector, currently in OCaml, is on the block to be rewritten as well.

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

These tools are currently in use internally. We plan to make them publicly available as part of our Freespire 2.0 release, scheduled for early 2007.

7.2 Haskell in Education

7.2.1 Functional programming at school

Report by: Walter Gussmann

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

Conditions at school

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

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

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

Haskell in tests

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

Teamwork with Haskell

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

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

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

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

What's new?

During the next weeks we will implement some fundamental cryptological algorithm. It starts with mono- and polyalphabetical encryption with a very easy implementation. Then we will implement the RSA-algorithm and some hash-functions.

All implementation should be tested with real textfiles although the key used are very, very weak.

What is coming in the future?

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

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

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

Further reading

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

7.3 Research Groups

7.3.1 Foundations and Methods Group at Trinity College Dublin

Report by: Andrew Butterfield
Participants: Andrew Butterfield, Glenn Strong, Hugh Gibbons, Yann Morvan

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

- Formal aspects of Functional I/O (→ 3.4.1)
- Using Testing to Debug Formal Models (→ 6.17)
- Supporting OO-Design technique for functional programmers (→ 3.3.7)
- Using functional programs as invariants in imperative programming

Members of other research groups at TCD have also used Haskell, such as the work done on Image rendering using GHC/OpenGL, in the Interaction, Simulation and Graphics Lab (→ 6.18).

Further reading

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

7.3.2 Foundations of Programming Group at the University of Nottingham

Report by: Liyang Hu *et al.*

The Nottingham FoP group is perhaps unique in the UK in bringing functional programming, type theory and category theory together to tackle fundamental issues in program construction. With a total of 28 people, we have a spectrum of interests:

Automated Reasoning Matthew Walton is exploring ways of exploiting automated reasoning techniques for dependently-typed programming languages such as Epigram, with a view to extend its verification capabilities. Current work is focused on implementating decision procedures as Epigram functions, and allowing the programmer to easily invoke said procedures.

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

Datatype-Generic Design Patterns Ondrej Rypacek together with Roland Backhouse and Henrik Nilsson are working on formal reasoning about object-oriented designs with emphasis on algebraic and datatype-generic methods. Our goal is a sound programming model expressive enough to capture object-oriented design patterns.

Dependently-Typed Haskell Supported by a Microsoft Research studentship, Robert Reitmeier is working on integrating dependent types in Haskell under the supervision of Thorsten Altenkirch, with advice

from Simon Peyton Jones. We are designing an alternative dependently-typed intermediate language, influenced by our experiences with Epigram.

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

Quantum Programming Thorsten Altenkirch, Jonathan Grattage and Alex Green are working on a Haskell-like quantum meta-language (QML), with quantum control as well as data structures. Guided by its categorical semantics, QML presents a constructive semantics of irreversible quantum computations. A Haskell implementation compiles QML into quantum circuits, giving it an operational semantics. A denotational semantics is given in terms of superoperators. We are investigating quantum IO for Haskell.

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

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

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

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

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

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

Yampa Yampa is an implementation of *functional reactive programming*, maintained by Henrik Nilsson. Some interesting discussions may be found on the yampa-users mailing list.

Teaching Haskell plays an important role in the undergraduate programme in Nottingham, via modules in Functional Programming, Advanced Functional Programming, Mathematics for Computer Science, Principles of Programming Languages, Compilers, and Computer-Aided Formal Verification, among others.

Programming in Haskell Graham Hutton has recently completed an introductory Haskell textbook (→1.5.2), to be published by Cambridge University Press before the end of 2006.

Future Events In Feburary, Nottingham will host the second Fun in the Afternoon: a termly seminar on functional programming and related topics. The aim is to have a few friendly and informal talks, as an antidote to the mid-term blues.

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

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

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

Further reading

- Foundations of Programming Group: http://cs.nott.ac.uk/Research/fop/
- Functional Programming at Nottingham: http://sneezy.cs.nott.ac.uk/fp/
- Epigram: http://e-pig.org/
- Quantum Programming: http://sneezy.cs.nott.ac.uk/qml/
- Yampa: http://haskell.org/yampa/
- Fun in the Afternoon: http://sneezy.cs.nott.ac.uk/fun/

- Midlands Graduate School 2007: http://cs.nott.ac.uk/MGS/
- FP Lunch: http://sneezy.cs.nott.ac.uk/fplunch/

7.3.3 Artificial Intelligence and Software Technology at JWG-University Frankfurt

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

DIAMOND

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

We introduced the *FUNDIO* calculus which proposes a non-standard way to combine lazy functional languages with I/O. A contextual equivalence depending on the Input/Output behavior of reduction sequences has been defined. A considerable set of program transformations has been shown to be correct. Moreover, we investigated several optimizations of evaluation, including strictness optimizations and an abstract machine, and have shown their correctness 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. After turning off 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 problematic optimizations 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 I/O 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 I/O operation wrapped by `unsafePerfomIO` does not matter in relation to all the other I/O operations of the program. So `unsafePerformIO` may be combined with monadic I/O in Haskell, and the result is reliable in the sense that I/O operations will not astonishingly be duplicated.

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

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

Haskell has a call-by-name semantics, but all efficient implementations of Haskell use call-by-need evaluation avoiding multiple evaluation of the same expression. Recently we showed equivalence of call-by-name and call-by-need for a tiny deterministic letrec-calculus and also the correctness of an unrestricted copy-reduction in both calculi. We expect our method scales up to extended letrec-calculi, for example extended by constructors and case-expressions.

Non-Deterministic Lambda-Calculi

Mutual Similarity In order to achieve more inference rules for equality in call-by-need lambda-calculi *Matthias Mann* has established a soundness (w.r.t. contextual equivalence) proof for mutual similarity in a non-deterministic call-by-need lambda calculus. Moreover, we have shown that Mann's approach scales up well to more expressive call-by-need non-deterministic lambda calculi, i.e. similarity can be used as a co-induction-based proof tool for establishing contextual preorder in a large class of untyped higher-order call-by-need calculi, in particular calculi with constructors, case, let, and non-deterministic choice. Current research is aimed towards extensions of these calculi towards Haskell.

Locally Bottom-Avoiding Choice We investigated an extended call-by-need lambda-calculus with a non-deterministic `amb`-operator together with a fair small-step reduction semantics. The appropriate program equivalence is contextual equivalence based on may- and must-termination. We proved that several program transformations preserve contextual equivalence, which permits useful program transformation, in particular partial evaluation using deterministic reductions. With the developed proof tools it appears promising to prove correctness of further program transformations. Future research should investigate also more involved inductive proof rules like Bird's take-lemma. A further challenge is to obtain a semantics preserving compiler for Haskell extended with `amb`.

Strictness Analysis using Abstract Reduction

The algorithm for strictness analysis using abstract reduction has been implemented at least twice: Once by Nöcker in C for Concurrent Clean and on the other hand by Schütz in Haskell in 1994. In 2005 we proved correctness of the algorithm by using a call-by-need lambda-calculus as a semantic basis.

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

Further reading

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

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

Report by:	Murray Gross

A grant has provided us with 6 new quad-processor machines, which we are currently integrating into our existing Linux/Mosix cluster. When the integration is complete, we will be comparing the performance and behavior of the Brooklyn College version of GpH (→ 3.2.1) and the SMP facility of the latest release of GHC (→ 2.1).

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

Contact

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

7.3.5 Functional Programming at Macquarie University

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

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

- We are close to releasing an alpha version of a port of the yhc (→ 2.4) runtime to Palm OS handhelds (→ 3.1.1).
- Kate Stefanov's thesis on off-the-shelf compression technology for bytecode-based programs is under examination.
- Matt Roberts continues to develop a simplified semantics for Barry Jay's pattern calculus that can be the basis of an efficient implementation.

Further reading

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

7.3.6 Functional Programming at the University of Kent

Report by:	Olaf Chitil

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

Our members pursue a variety of Haskell-related projects, many of which are reported in other sections of this report.

The refactoring team Huiqing Li, Simon Thompson, Chris Brown and Claus Reinke have released further snapshots of HaRe, the Haskell Refactorer (→ 5.3.2) and are now refactoring Erlang programs. Huiqing Li recently published her PhD thesis on refactoring Haskell programs. The team's work on refactoring Erlang is strengthened by two students from Budapest, Aniko Vig and Tamas Nagy, who are visiting for four months. Nik Sultana has joined the group as an MSc student working with Simon on formal proofs of Haskell programs. Thomas Davie, Yong Luo and Olaf Chitil are working together with the York functional programming group on developing the Haskell tracer Hat (→ 5.4.2) further. They are looking in particular at extensions and improvements of algorithmic debugging. Axel Simon maintains the gtk2hs binding to the Gtk+ GUI library (→ 4.8.2) in cooperation with Duncan Coutts, Oxford University. Keith Hanna is continuing work on Vital, a document-centered programming environment for Haskell, and on Pivotal (→ 3.1.2), a GHC-based implementation of a similar environment.

Further reading

- FP group:
  http://www.cs.kent.ac.uk/research/groups/tcs/fp/
- Refactoring Functional Programs:
  http://www.cs.kent.ac.uk/projects/refactor-fp/
- Refactoring Haskell programs. Huiqing Li. PhD thesis, Computing Laboratory, University of Kent, Canterbury, Kent, UK, September 2006.

- Hat:
  http://www.haskell.org/hat/
- Gtk2HS:
  http://www.haskell.org/gtk2hs
- Vital:
  http://www.cs.kent.ac.uk/projects/vital/
- Pivotal:
  http://www.cs.kent.ac.uk/projects/pivotal/

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

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

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

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

Languages

The group has designed, implemented, evaluated and used several high performance/distributed functional languages, and continues to do so. High performance languages include Glasgow parallel Haskell (→ 3.2.1) and Parallel ML with skeletons (PMLS). Distributed/mobile languages include Glasgow distributed Haskell (→ 3.2.2), Erlang (http://www.erlang.org/), Hume (http://www-fp.dcs.st-and.ac.uk/hume/), JoCaml, Camelot, Java Voyager and Java Go.

Projects

Current projects include

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

Collaborations

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

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

Further reading

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

7.3.8 Programming Languages & Systems at UNSW

Report by:	Manuel Chakravarty

The PLS research group at the University of New South Wales, Sydney, has produced a couple of Haskell tools and libraries, including the new high-performance packed string library `Data.ByteString` (→ 4.6.3), the hs-plugins (→ 4.4.1) library for dynamically loaded type-safe plugins, the interface generator C→Haskell (→ 5.1.2), and the dynamic editor Yi (→ 6.9).

In cooperation with GHC HQ at Microsoft Research, Cambridge, we introduced the idea of type classes with *associated types*, and with GHC HQ and Martin Sulzmann, from the National University of Singapore, we proposed GHC's new intermediate language System F_C. System F_C improves GHC's Core intermediate language to support the unified implementation of guarded abstract data types, functional dependencies, and associated types, while simultaneously broadening the range of programs that GHC can translate. F_C is fully implemented in GHC's current development version 6.7, along with a modest extension to our proposal on associated data types; see http://haskell.org/haskellwiki/GHC/Indexed_types for details.

Together with GHC HQ, we are busy with finally bringing nested data parallelism to GHC, with a focus to utilise multi-core CPUs. We summarised our first results in a recent paper available from http://www.cse.unsw.edu.au/~chak/papers/CLPKM06.html.

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

7.4 User groups

7.4.1 Fedora Haskell

Report by: Jens Petersen

Fedora Haskell provides packages of certain Haskell projects for Fedora Core in yum repositories. The main news is that hugs98 (→ 2.2) and gtk2hs (→ 4.8.2) have been added to in Fedora Extras (thanks to Gérard Milmeister). Also ghc (→ 2.1) was updated to 6.4.2 and darcs (→ 6.4) to 1.0.7. I hope more Haskell packages submitted and accepted in Extras in the coming period. There is a mailing list ⟨fedora-haskell@haskell.org⟩ for announcements and questions. Contributions are needed, particular in the form of submissions and reviewing of packages for Fedora Extras.

Further reading

http://haskell.org/fedora/

7.4.2 OpenBSD Haskell

Report by: Don Stewart

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

GHC (→ 2.1) is available for i386 and amd64. nhc98 (→ 2.3) is available for i386 and sparc. Hugs (→ 2.2) is available for the alpha, amd64, hppa, i386, powerpc, sparc and sparc64. A number of other Haskell tools and libraries are also available, including alex (→ 5.2.2), happy (→ 5.2.3), haddock (→ 5.5.6) and darcs (→ 6.4).

Support for the GHC head branch continues, recent bugs relating to dynamic linking and ghci have been fixed.

7.4.3 Haskell in Gentoo Linux

Report by: Andres Löh

Lennart Kolmodin is a new member of the Gentoo Haskell team, which is now five persons strong.

Recent work has focused on the transition to ghc-6.6. There are some tricky problems involved supporting both ghc-6.4.* and ghc-6.6 with the modular libraries in Gentoo's package manager, but we hope to have things running before the end of the year.

You can access and test the latest versions of the ebuilds we are working on via our darcs (→ 6.4) overlay, which is now also available via the Gentoo overlay manager "layman". Please report problems with the overlay on IRC (`#gentoo-haskell` on freenode), where we coordinate development.

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

7.5 Individuals

7.5.1 Oleg's Mini tutorials and assorted small projects

Report by: Oleg Kiselyov

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

Reversing Haskell typechecker: converting from `undefined` to defined

We demonstrate how to make the Haskell typechecker work in reverse: to infer a term of a given type. For example:

```
rtest4 f g =
  rr (undefined::(b -> c) -> (a -> b) -> a -> c)
     HNil f g
*HC> rtest4 (:[]) Just 'x'
[Just 'x']
*HC> rtest4 Just Right True
Just (Right True)
```

We ask the Haskell typechecker to derive us a function of the specified type. We get the real function, which we can then apply to various arguments. The return result does behave like a 'composition' – which is what the type specifies. Informally, we converted from *undefined* to defined. We can also print the inferred term – which can be useful, for example, for converting from

a point-less to the more comprehensible point-full representation.

Our system solves type habitation for a class of functions with polymorphic types. From another point of view, the system is a prover in the implication fragment of intuitionistic logic. Essentially we turn a *type* into a logical program – a set of Horn clauses – which we then solve by SLD resolution. It is gratifying to see that Haskell typeclasses are up to that task.

Compared to a similar system Djinn, our system is not a separate stand-alone application. Rather, our de-typechecker is the regular Haskell function, whose result (a functional value of the desired type) can be immediately used in the rest of the program. That is, the term 'generation' is done by the Haskell typechecker itself rather in a separate (meta-) application.

Further Reading

- Inverse typechecking and theorem proving in intuitionistic and classical logics: http://pobox.com/~oleg/ftp/Computation/Computation.html#typechecker-CH
- http://pobox.com/~oleg/ftp/Haskell/types.html#de-typechecker

How to zip folds: A library of fold transformers (streams)

We present a library of potentially infinite "lists" realized as folds (aka streams, aka success-failure-continuation–based generators). Whereas the standard Haskell Prelude functions such as `map` and `take` transform lists, we transform folds. We implement the range of progressively more complex transformers – from `map`, `filter`, `takeWhile` to `take`, to `drop` and `dropWhile`, and finally, `zip` and `zipWith`. The latter shows merging of two folds 'elementwise'.

Emphatically we never convert a stream to a list and so we *never use recursion or recursive types*. All iterative processing is driven by the fold itself. We only need higher-ranked types, because lists cannot be fully implemented in a simply-typed lambda-calculus.

The implementation of zip also solves the problem of "parallel loops". One can think of a fold as an accumulating loop and realize a nested loop as a nested fold. Representing a parallel loop as a fold is a challenge, answered at the end of the mini-tutorial.

http://pobox.com/~oleg/ftp/Algorithms.html#zip-folds

Representing knowledge about knowledge: "Mr.S and Mr.P" puzzle

We describe a *concise* Haskell solution to the "Mr.S and Mr.P" puzzle posed by John McCarthy (Formalization of two Puzzles Involving Knowledge. 1987. http://www-formal.stanford.edu/jmc/puzzles.html). We rely on the straightforward encoding of multiple-world semantics of modalities. Our Haskell code demonstrates a generic method of encoding facts, and the knowledge about facts, and the knowledge of the knowledge, etc. Incidentally, compared to the notation in McCarthy's paper, the Haskell notation is notably concise.

http://pobox.com/~oleg/ftp/Algorithms.html#mr-s-p

Lightweight dependent typing: eliminating array bound check

The previously mentioned mini-tutorial on Lightweight dependent typing has a new example: Knuth-Morris-Pratt (packed) string matching. The code faithfully implements the original (imperative) algorithm, using PackedStrings and STArrays. The algorithm also uses indices in a creative way, including a deliberately out-of-bounds index -1. Yet our implementation statically guarantees the safety of all array/string access operations and removes the need for index bound checks. Thus our code is both fast and safe.

http://pobox.com/~oleg/ftp/Computation/lightweight-dependent-typing.html

7.5.2 Implementation of "How to write a financial contract"

Report by: Alain Crémieux

The aim is to produce a reference implementation of "Composing contracts: an adventure in financial engineering" (http://research.microsoft.com/Users/simonpj/#contracts-icfp), which could be used as a basis for implementing other DSELs. At present the implementation is divided in 5 layers, from "basic" to "optimizing". Now that GADTs are supported in GHC (→ 2.1), it is possible to express a tagless interpreter for the contract language in a very concise way, even if it is still necessary to guide the typechecker with some annotations. So the next step is to use Omega, where these annotations are not necessary thanks to the possibility of defining named kinds. And to generalize the contract language to some typed lambda-calculus, including staging. With this I can obtain an optimised interpreter, valuating correctly financial options (the result is easy to check w.r.t. financial books).

Code available on demand.

7.5.3 Inductive Programming

Report by: Lloyd Allison

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

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

A student project by J. Bardsley (see below) used Template-Haskell to automate the definition of data-handling routines, types, and some type-class instance declarations, as required to analyse a given multivariate data-set.

A case-study defines a learner for the structure and the parameters of Bayesian networks over mixed variables (data attributes): discrete, continuous, and even structured variables; the learner was applied to a Search and Rescue data-set on missing people. This data-set has many missing values which gives great scope for bad puns.

IP has also been used to analyse ecological transects[1], and mutation data on a drug-resistant virus, two applications where IP's flexibility is very useful.

Other case-studies include mixtures of time-series, Bayesian networks, and time-series models and "the" sequence-alignment dynamic-programming algorithm. Currently there are types and classes for models (various probability distributions), function-models (regressions), time-series (e.g. Markov models), mixture models, and classification trees (plus regression trees and model trees). A spring-clean of the code is long overdue.

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

Future plans

Planned are more applications to real data-sets, and comparisons against other learners. A big rewrite will happen, one day.

Further reading

- M. B. Dale, L. Allison, P. E. R. Dale. Segmentation and Clustering as Complementary Sources of Information. Acta Oecologica, to appear.
- J. Bardsley. Generalising Data Description for Machine Learning, 2006. http://www.csse.monash.edu.au/hons/projects/2006/James.Bardsley
- L. Allison. A Programming Paradigm for Machine Learning with a Case Study of Bayesian Networks. ACSC, pages 103–111, January 2006. http://crpit.com/confpapers/CRPITV48Allison.pdf
- Other reading is listed at the URL: http://www.csse.monash.edu.au/~lloyd/tildeFP/II/

7.5.4 Bioinformatics tools

Report by:	Ketil Malde

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

Everything is – of course – available as darcs repos (→ 6.4), at
http://www.ii.uib.no/~ketil/bioinformatics/repos.

Further reading

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

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

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

I'm a mathematician, with expertise in dynamical systems and probability. I'm using math to model language acquisition, variation, and change. My current project is about testing various hypotheses put forth by the linguistics community concerning the word order of English. Old and Middle English had significantly different syntax than Modern English, and the development of English syntax is perhaps the best studied case of language change in the world. My overall plan is to build simulations of various stages of English and test them against manuscript data, such as the Pennsylvania Parsed Corpus of Middle English (PPCME).

One of my projects is 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.

I'm also experimenting with GSLHS. I'm using it to study a linear reward-penalty learning algorithm and a new algorithm based on a differential equation.

I use GHC and Hugs on Fedora Linux (→ 7.4.1).

I'm also working on an interpreted language called Crouton. It's based very loosely on Haskell's syntax

and lazy evaluation, but without the type system and with much more powerful pattern matching. It will allow me to scan files from the PPCME and other corpora in lisp-like formats, find particular constructions, and transform them. Patterns can be as complex as context free grammars, and apply to whole structures as well as strings. I expect it to be a big help in the data collection part of my language modeling.

Further reading

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