Haskell Communities and Activities Report

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

Twelfth Edition – May 30, 2007

Andres Löh (ed.)
Lloyd Allison, Tiago Miguel Laureano Alves, Krasimir Angelov
Carlos Areces, Alistair Bayley, Jean-Philippe Bernardy
Clifford Beshers, Chris Brown, Bjorn Buckwalter
Andrew Butterfield, Manuel Chakravarty, Olaf Chitil
Duncan Coutts, Jácome Cunha, Atze Dijkstra
Frederik Eaton, Martin Erwig, Jeroen Fokker
Richard A. Frost, Clemens Fruhwirth, Andy Gill
Dimitry Golubovsky, Daniel Gorin, Martin Grabmüller
Murray Gross, Walter Gussmann, Kevin Hammond
Christopher Lane Hinson, Guillaume Hoffmann, Paul Hudak
Liyang Hu, Graham Hutton, S. Alexander Jacobson
Wolfgang Jeltsch, Antti-Juhani Kaijanaho, Jeremy O'Donoghue
Oleg Kiselyov, Dirk Kleeblatt, Lennart Kolmodin
Slawomir Kolodynski, Eric Kow, Huiqing Li
Andres Löh, Rita Loogen, Salvador Lucas
Ian Lynagh, Ketil Malde, Christian Maeder
Simon Marlow, Conor McBride, Arie Middelkoop
Neil Mitchell, William Garret Mitchener, Andy Adams-Moran
Dino Morelli, Yann Morvan, Diego Navarro
Rishiyur Nikhil, Stefan O'Rear, Sven Panne
Ross Paterson, Simon Peyton-Jones, Claus Reinke
Colin Runciman, Alberto Ruiz, David Sabel
Uwe Schmidt, Alexandra Silva, Ganesh Sittampalam
Anthony Sloane, Dominic Steinitz, Donald Bruce Stewart
Jennifer Streb, Glenn Strong, Martin Sulzmann
Doaitse Swierstra, Wouter Swierstra, Hans van Thiel
Henning Thielemann, Peter Thiemann, Simon Thompson
Phil Trinder, Miguel Vilaca, Joost Visser
Edsko de Vries, Malcolm Wallace, Mark Wassell
Stefan Wehr, Ashley Yakeley, Bulat Ziganshin

Preface

You are reading the twelfth edition of the Haskell Communities and Activities Report – as always, containing entries from enthusiastic Haskellers all over the world.

This edition has 138 entries, 33 of them are completely new (and therefore highlighted with a blue background), and 54 have had updates since the previous edition (and have a header with a blue background). All entries that have not been updated for a year or longer have been removed to make sure that your are reading information that is as up-to-date as possible.

I want to use the opportunity to thank all the contributors. This report has 90 authors, but the number of total contributors to all the projects reported on is much, much greater. I find it wonderful that the Haskell communities continue to be so diverse and open at the same time.

As always, I want to encourage you to watch out for projects that are missing from this report, and to make their authors aware of the report so that they can contribute to the November edition (deadline probably around the end of October).

Feedback is very welcome at ⟨hcar@haskell.org⟩. Pleasant reading!

Andres Löh, University of Bonn, Germany

Contents

1 General **7**
1.1 HaskellWiki and haskell.org **7**
1.2 `#haskell` **7**
1.3 Planet Haskell **7**
1.4 Haskell Weekly News **8**
1.5 The Monad.Reader **8**
1.6 Books and tutorials **8**
1.6.1 New textbook – Programming in Haskell 8
1.6.2 Haskell Wikibook (was: Haskell Tutorial Wikibook) 9
1.6.3 Haskell Tutorials in Portuguese 9
1.7 A Survey on the Use of Haskell in Natural-Language Processing **9**

2 Implementations **10**
2.1 The Glasgow Haskell Compiler **10**
2.2 Hugs **12**
2.3 nhc98 **12**
2.4 yhc **13**

3 Language **14**
3.1 Variations of Haskell **14**
3.1.1 Liskell 14
3.1.2 Haskell on handheld devices 14
3.1.3 Camila 14
3.2 Non-sequential Programming **15**
3.2.1 GpH – Glasgow Parallel Haskell 15
3.2.2 Eden 16
3.3 Type System/Program Analysis **16**
3.3.1 Epigram 16
3.3.2 Chameleon project 17
3.3.3 XHaskell project 18
3.3.4 ADOM: Agent Domain of Monads 18
3.3.5 EHC, ‘Essential Haskell’ Compiler 18
3.3.6 Uniqueness Typing 19
3.3.7 Uniqueness Typing in EHC 19
3.3.8 Object-Oriented Haskell 20
3.4 IO **20**
3.4.1 Formal Aspects of Pure Functional I/O 20

4 Libraries **21**
4.1 Packaging and Distribution **21**
4.1.1 Core 21
4.2 General libraries **21**
4.2.1 Test.IOSpec 21
4.2.2 PFP – Probabilistic Functional Programming Library for Haskell 21
4.2.3 GSLHaskell 22
4.2.4 An Index Aware Linear Algebra Library 22
4.2.5 Haskell Rules: Embedding Rule Systems in Haskell 22
4.3 Parsing and transforming **22**
4.3.1 InterpreterLib 22
4.3.2 hscolour 23
4.3.3 Utrecht Parsing Library and Attribute Grammar System 23
4.3.4 Left-Recursive Parser Combinators 23

4.3.5 RecLib – A Recursion and Traversal Library for Haskell . . . 24
4.4 System . . . 24
4.4.1 Harpy . . . 24
4.4.2 hs-plugins . . . 24
4.4.3 The libpcap Binding . . . 25
4.4.4 Streams . . . 25
4.4.5 System.FilePath . . . 25
4.4.6 hinotify . . . 25
4.5 Databases and data storage . . . 26
4.5.1 CoddFish . . . 26
4.5.2 Takusen . . . 26
4.6 Data types and data structures . . . 27
4.6.1 Standard Collection Libraries . . . 27
4.6.2 Data.ByteString . . . 27
4.6.3 Data.List.Stream . . . 27
4.6.4 dimensional . . . 27
4.6.5 Numeric prelude . . . 28
4.6.6 HList – a library for typed heterogeneous collections . . . 28
4.6.7 ArrayRef . . . 29
4.7 Data processing . . . 29
4.7.1 AltBinary . . . 29
4.7.2 binary . . . 30
4.7.3 binarydefer . . . 30
4.7.4 Compression-2006 (was: Compression-2005) . . . 30
4.7.5 The Haskell Cryptographic Library . . . 31
4.7.6 The Haskell ASN.1 Library . . . 31
4.7.7 2LT: Two-Level Transformation . . . 31
4.8 User interfaces . . . 32
4.8.1 Grapefruit – A declarative GUI library . . . 32
4.8.2 wxHaskell . . . 32
4.8.3 Gtk2Hs . . . 33
4.8.4 hscurses . . . 33
4.8.5 VTY . . . 33
4.9 (Multi-)Media . . . 34
4.9.1 HOpenGL – A Haskell Binding for OpenGL and GLUT . . . 34
4.9.2 HOpenAL – A Haskell Binding for OpenAL and ALUT . . . 34
4.9.3 Haskore revision . . . 34
4.10 Web and XML programming . . . 35
4.10.1 HAppS – Haskell Application Server . . . 35
4.10.2 Pass.Net . . . 36
4.10.3 Converter of Yhc Core to Javascript (ycr2js) . . . 36
4.10.4 tagsoup . . . 37
4.10.5 HaXml . . . 37
4.10.6 Haskell XML Toolbox . . . 37
4.10.7 WASH/CGI – Web Authoring System for Haskell . . . 38

5 Tools 39
5.1 Foreign Function Interfacing . . . 39
5.1.1 C→Haskell . . . 39
5.2 Scanning, Parsing, Analysis . . . 39
5.2.1 Alex version 2 . . . 39
5.2.2 Happy . . . 39
5.2.3 SdfMetz . . . 39
5.2.4 XsdMetz: metrics for XML Schema . . . 40
5.3 Transformations . . . 40
5.3.1 derive . . . 40
5.3.2 Term Rewriting Tools written in Haskell . . . 40
5.3.3 HaRe – The Haskell Refactorer . . . 41

5.3.4 VooDooM . . . 42
5.4 Testing and Debugging . . . **42**
5.4.1 Haskell Program Coverage . . . 42
5.4.2 Hat . . . 43
5.4.3 SmallCheck: another lightweight testing library in Haskell . . . 43
5.5 Development . . . **44**
5.5.1 hmake . . . 44
5.5.2 Haskell Modes for Vim . . . 44
5.5.3 Ruler . . . 44
5.5.4 cpphs . . . 45
5.5.5 Visual Haskell . . . 45
5.5.6 Haddock . . . 45
5.5.7 Hoogle – Haskell API Search . . . 45
5.5.8 SearchPath . . . 46

6 Applications **47**
6.1 xmonad . . . **47**
6.2 GenI . . . **47**
6.3 Roguestar . . . **47**
6.4 mmisar . . . **47**
6.5 Inference Services for Hybrid Logics . . . **48**
6.5.1 HyLoRes . . . 48
6.5.2 HTab . . . 48
6.5.3 HGen . . . 48
6.6 Raskell . . . **48**
6.7 photoname . . . **49**
6.8 HJS – Haskell Javascript Interpreter . . . **49**
6.9 FreeArc . . . **49**
6.10 Darcs . . . **50**
6.11 lambdabot . . . **50**
6.12 yi . . . **50**
6.13 INblobs – Interaction Nets interpreter . . . **51**
6.14 lhs2TEX . . . **51**
6.15 Emping . . . **51**
6.16 Audio signal processing . . . **51**
6.17 hmp3 . . . **52**
6.18 Testing Handel-C Semantics Using QuickCheck . . . **52**
6.19 easyVision . . . **52**
6.20 View selection for image-based rendering . . . **53**

7 Users **54**
7.1 Commercial users . . . **54**
7.1.1 Credit Suisse Global Modelling and Analytics Group . . . 54
7.1.2 Bluespec tools for design of complex chips . . . 54
7.1.3 Galois, Inc. . . . 55
7.1.4 Linspire . . . 56
7.2 Haskell in Education . . . **56**
7.2.1 Functional programming at school . . . 56
7.3 Research Groups . . . **57**
7.3.1 Foundations and Methods Group at Trinity College Dublin . . . 57
7.3.2 Foundations of Programming Group at the University of Nottingham . . . 57
7.3.3 Artificial Intelligence and Software Technology at JWG-University Frankfurt . . . 59
7.3.4 Formal Methods at Bremen University and DFKI Lab Bremen . . . 60
7.3.5 Functional Programming at Macquarie University . . . 60
7.3.6 Functional Programming at the University of Kent . . . 60
7.3.7 Programming Languages & Systems at UNSW . . . 61
7.3.8 Haskell in Romania . . . 61
7.3.9 SCIence project . . . 62

7.4 User groups . . . **62**
7.4.1 OpenBSD Haskell . . . 62
7.4.2 Haskell in Gentoo Linux . . . 62
7.5 Individuals . . . **62**
7.5.1 Oleg's Mini tutorials and assorted small projects . . . 62
7.5.2 Inductive Programming . . . 63
7.5.3 Bioinformatics tools . . . 64
7.5.4 Using Haskell to implement simulations of language acquisition, variation, and change . . . 64

1 General

1.1 HaskellWiki and haskell.org

Report by:	Ashley Yakeley

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

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

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

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

Further reading

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

1.2 `#haskell`

Report by:	Don Stewart

The `#haskell` IRC channel is a real-time text chat where anyone can join to discuss Haskell. The channel has grown dramatically in users over the last 6 months, and now `#haskell` averages over 300 concurrent users (with a high water mark of 340 users), and is one of the biggest channels on freenode. The irc channel is home to hpaste and lambdabot, two useful Haskell bots. 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.dut` – Dutch speakers
- `#haskell.es` – Spanish speakers
- `#haskell.fi` – Finnish speakers
- `#haskell.fr` – French speakers
- `#haskell.hr` – Croatian speakers
- `#haskell.it` – Italian speakers
- `#haskell.jp` – Japenese speakers
- `#haskell.no` – Norwegian speakers
- `#haskell_ru` – Russian speakers
- `#haskell.se` – Swedish speakers

Related Haskell channels are now emerging, including:

- `#haskell-overflow` – Overflow conversations
- `#haskell-blah` – Haskell people talking about anything except Haskell itself
- `#gentoo-haskell` – Gentoo/Linux specific Haskell conversations (→ 7.4.2)
- `#darcs` – Darcs revision control channel (written in Haskell) (→ 6.10)
- `#ghc` – GHC developer discussion (→ 2.1)
- `#happs` – HAppS Haskell Application Server channel (→ 4.10.1)
- `#xmonad` – Xmonad a tiling window manager written in Haskell (→ 6.1)

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. (→ 7.1.3) 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, jobs, 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).

Further reading

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

1.5 The Monad.Reader

Report by:	Wouter Swierstra

There are plenty of academic papers about Haskell and plenty of informative pages on the Haskell Wiki. Unfortunately, there's not much between the two extremes. That's where The Monad.Reader tries to fit in: more formal than a Wiki page, but more casual than a journal article.

There are plenty of interesting ideas that maybe don't warrant an academic publication – but that doesn't mean these ideas aren't worth writing about! Communicating ideas to a wide audience is much more important than concealing them in some esoteric journal. Even if its all been done before in the Journal of Impossibly Complicated Theoretical Stuff, explaining a neat idea about 'warm fuzzy things' to the rest of us can still be plain fun.

The Monad.Reader is also a great place to write about a tool or application that deserves more attention. Most programmers don't enjoy writing manuals; writing a tutorial for The Monad.Reader, however, is an excellent way to put your code in the limelight and reach hundreds of potential users.

I do try to publish a new issue quarterly, but I'm completely reliant on your submissions. So please consider contributing to the functional programming community by writing something for The Monad.Reader!

Further reading

All the recent issues and the information you need to start writing an article are available from: http://www.haskell.org/haskellwiki/The_Monad.Reader.

1.6 Books and tutorials

1.6.1 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:

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

Further information:

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

1.6.2 Haskell Wikibook (was: Haskell Tutorial Wikibook)

Report by:	Eric Kow
Participants:	Apfelmus
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 added some original content, giving a friendly introduction to advanced topics: Category Theory, Denotational Semantics, The Curry-Howard isomorphism and Zippers. Thanks to David House and Apfelmus for their hard work and to the Haskell community for your helpful comments! (Of course, one of our greatest dreams is a module by one of the very founders of the Haskell programming language.)

The wikibook is starting to be recognised as a useful resource for beginners in Haskell, and has been receiving some positive comments from the blogosphere. The wikibook has even selected for inclusion into the list of "featured books" on the English wikibooks project. It will now be prominently displayed on the wikibooks front page in rotation with other featured books.

Our community has been starting to grow, in the meantime. For example, the Polish and Russian Haskell wikibooks have been rather active in the last six months. Want to see a Haskell wikibook in your language? Be bold and get started! While you're at it, you might even consider participating in our new mailing list, ⟨wikibook@haskell.org⟩.

Further reading

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

1.6.3 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/

1.7 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

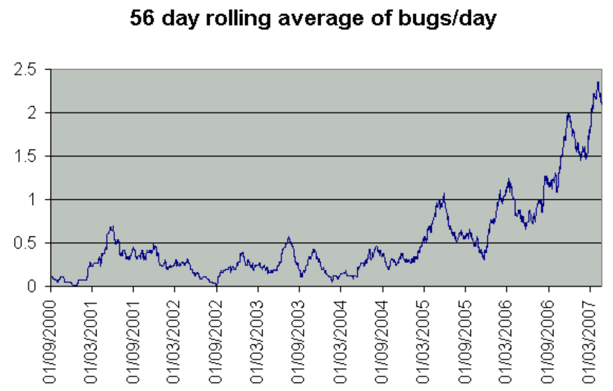
A draft of the survey is available at:
http://cs.uwindsor.ca/~richard/PUBLICATIONS/NLI_LFP_SURVEY_DRAFT.pdf

2 Implementations

2.1 The Glasgow Haskell Compiler

Report by: Simon Peyton-Jones, Simon Marlow, Ian Lynagh

GHC continues to thrive. One indicator of how widely GHC is used is the number of bug reports we get. Here is a graph showing how the number of bug reports filed has varied with time:



You could interpret these figures as saying that GHC is getting steadily more unreliable! But we don't think so ... we believe that it's mostly a result of more people using GHC, for more applications, on more platforms.

As well as more bug reports, we are getting more help from the community, too. Some people regularly commit patches, and we get a steady trickle of patches emailed in from folk who (mostly) do not have commit rights, but who have built GHC, debugged a problem, sent us the patch. Our thanks go out to Aaron Tomb, Alec Berryman, Alexey Rodriguez, Andrew Pimlott, Andy Gill, Bas van Dijk, Bernie Pope, Bjorn Bringert, Brian Alliet, Brian Smith, Chris Rodrigues, Claus Reinke, David Himmelstrup, David Waern, Judah Jacobson, Isaac Jones, Lennart Augustsson, Lennart Kolmodin, Manuel M T Chakravarty, Pepe Iborra, Ravi Nanavati, Samuel Bronson, Sigbjorn Finne, Spencer Janssen, Sven Panne, Tim Chevalier, Tim Harris, Tyson Whitehead, Wolfgang Thaller, and anyone else who has contributed but we have accidentally omitted.

As a result of this heavy usage, it has taken us nearly six months to stabilise GHC 6.6.1, fixing over 100 reported bugs or infelicities in the already-fairly-solid GHC 6.6.

The HEAD (which will become GHC 6.8) embodies nine months of development work since we forked the tree for GHC 6.6. We are now aiming to get a stable set of features implemented in the HEAD, with a view to forking off the GHC 6.8 branch in the early summer. As our last HCAR report indicated, there will be lots of new stuff in GHC 6.8. The rest of this entry describes the features that are likely to end up in 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/).

Simon Peyton Jones, Simon Marlow, Ian Lynagh

Type system and front end

- 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 "System F with Type Equality Coercions" (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 "data-type families" (aka indexed data types), a modest generalisation of the "associated data types" of our POPL'05 paper "Associated types with class" (http://research.microsoft.com/~simonpj/papers/assoc-types/)

  This part is done. Now we are working on "type-synonym families" (aka type functions or associated type synonyms (ICFP'05) (http://research.microsoft.com/~simonpj/papers/assoc-types), which are considerably trickier that data type families, at least so far as type inference is concerned. Tom Schrijvers is in Cambridge for three months to help us use ides from Constraint Handling Rules to solve the inference problem. Type synonym families will almost completely fill the spot occupied by the always-troublesome functional dependencies, so we are quite excited about this.

  Details are at http://haskell.org/haskellwiki/GHC/Indexed_types.

- Simon PJ finally implemented "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 are now finally fixed. Implication constraints are described by Martin

Sulzmann in "A framework for Extended Algebraic Data Types" (http://www.comp.nus.edu.sg/~sulzmann/publications/tr-eadt.ps.gz).

- 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://haskell.org/haskellwiki/GHC/StandAloneDeriving.

- Lennart Augustsson implemented overloaded string literals. So now just as a numeric literal has type `forall a. Num a => a`, so a string literal has type `forall a. IsString a => a`, The documentation is here: http://www.haskell.org/ghc/dist/current/docs/users_guide/other-type-extensions.html#overloaded-strings.

A less successful feature of the last year has been the story on impredicative instantiation (see the paper "Boxy types: type inference for higher-rank types and impredicativity" (http://research.microsoft.com/~simonpj/papers/boxy). The feature is implemented, but the implementation is significantly more complicated than we expected; and it delivers fewer benefits than we hoped. For example, the system described in the paper does not type-check (`runST $ foo`) and everyone complains. So Simon PJ added an even more ad-hoc extension that does left-to-right instantiation.

The power-to-weight ratio is not good. We're still hoping that Dimitrios Vytiniotis and Stephanie Weirich will come out with a simpler system, even if it's a bit less powerful. So don't get too used to impredicative instantiation as it now stands; it might change!

Optimisations

- Simon PJ rewrote the Simplifier (again). It isn't clear whether it was that alone, or whether something else happened too, but performance has improved quite significantly; on the order of 12%.

- Roman Leshchinskiy, Don Stewart, and Duncan Coutts did some beautiful work on "fusion"; see their paper "Rewriting Haskell strings" (http://www.cse.unsw.edu.au/~dons/papers/CSL06.html). This fusion work is already being heavily used in the parallel array library (see below), and they are also working on replacing foldr/build fusion with stream fusion in the main base library (→ 4.6.2) (→ 4.6.3).

  Their work highlighted the importance of the SpecConstr transformation, which Simon PJ implemented several years ago. Of course, they suggested many enhancements, many of which Simon PJ duly implemented; see the new paper "Constructor specialisation for Haskell programs" (http://research.microsoft.com/~simonpj/papers/spec-constr/).

- Alexey Rodriguez visited us for three months from Utrecht, and implemented a new back-end optimisation called "dynamic pointer tagging". We have wanted to do this for ages, but it needed a skilled and insightful hacker to make it all happen, and Alexey is just that. This optimisation alone buys us another 15% performance for compiled programs: see the paper "Dynamic pointer tagging" (http://research.microsoft.com/~simonpj/papers/ptr-tag/index.htm).

Concurrency

- Gabriele Keller, Manuel Chakravarty, and Roman Leshchinskiy, at the University of New South Wales, are collaborating with us on support for "nested data-parallel computation" in GHC. We presented a paper "Data parallel Haskell: a status report" (http://research.microsoft.com/~simonpj/papers/ndp) at the Declarative Aspects of Multicore Programing workshop in January 2007, and made a first release of the library in March. It's a pretty ambitious project, and we have quite a way to go. You can peek at the current status on the project home page: http://haskell.org/haskellwiki/GHC/Data_Parallel_Haskell.

- Tim Harris added support for "invariants" to GHC's Software Transactional Memory (STM) implementation. Paper is "Transactional memory with data invariants" (http://research.microsoft.com/~simonpj/papers/stm/).

- 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). It works! But alas, doing GC with two processors runs no faster than with one! (We do plan to investigate this further and find the source of the bottleneck.)

Peng Li, from the University of Pennsylvania, spent an exciting three months at Cambridge, working on a whole new architecture for concurrency in GHC. (If you don't know Peng you should read his wonderful paper "Combining Events And Threads For Scalable Network Services" (http://www.seas.upenn.edu/~lipeng/homepage/papers/lz07pldi.pdf) on implementing a network protocol stack in Haskell.) At the moment GHC's has threads, scheduling, `forkIO`, `MVars`, transactional memory, and more besides, all "baked into" the run-time system and implemented in C. If you want to change this implementation you have to either be Simon Marlow, or else very brave indeed. With Peng (and help from Andrew Tolmach, Olin Shivers, Norman Ramsey) we designed a new, much lower-level set of primitives, that should allow us to implement all of the above "in Haskell". If you want a different scheduler, just code it up in Haskell, and plug it in.

Peng has a prototype running, but it has to jump the "Marlow barrier" of being virtually as fast as the existing C runtime; so far we have not committed to including this in GHC, and it certainly won't be in GHC 6.8. No paper yet, but look out for a Haskell Workshop 2007 submission.

Programming environment

There have been some big developments in the programming environment:

- Andy Gill implemented the Haskell Program Coverage (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.)

- Pepe Iborra, Bernie Pope, and Simon Marlow have leveraged the same "tick" points used in the Haskell Program Coverage work to implement a breakpoint debugger in GHCi http://hackage.haskell.org/trac/ghc/wiki/NewGhciDebugger. Unlike HAT, which transforms the whole program into a new program that generates its own (massive) trace, this is a cheap-and-cheerful debugger. It simply lets you set breakpoints and look around to see what is in the heap, more in the manner of a conventional debugger. No need to recompile your program: it "just works".

- Aaron Tomb and Tim Chevalier are working on resurrecting External Core, whose implementation was not only bit-rotted, but also poorly designed (by Simon PJ). By GHC 6.8 we hope to be able to spit out External Core for any program, perhaps transform it in some external program, and read it in again, surviving the round trip unscathed.

- It is now possible to compile to object code instead of bytecode inside GHCi, simply by setting a flag (`-fobject-code`).

- The GHC API has seen some cleanup, and it should now be both more complete and slightly easier to use. There is still plenty of work to do here, though.

- David Waern has been working on integrating Haddock and GHC during his Google Summer of Code project last year. The parts of this project that involved modifying GHC are done and integrated into the GHC tree. The new version of Haddock based on GHC is usable but still experimental; the darcs repository is http://darcs.haskell.org/SoC/haddock.ghc.

Libraries

- The set of "corelibs" has been further streamlined, with `parsec`, `regex-base`, `regex-compat`, `regex-posix` and `stm` moved to extralibs in the HEAD. This disentangles releases of these packages from the GHC release process, and also means that development builds of GHC are quicker as they don't need to build those libraries.

- We plan to extract parts of the base package into separate smaller packages; see http://www.haskell.org/pipermail/libraries/2007-April/007342.html on the libraries mailing list.

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 much 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 we have focused on integrating the standard haskell.org libraries (we have gained Data.Map and others) – but still have some way to go. We have also enhanced our Yhc.Core library, gaining many new users, and have produced an article for The Monad.Reader (→ 1.5) on the applications of Yhc.Core.

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 Liskell

Report by:	Clemens Fruhwirth
Status:	experimental

When Haskell consists of Haskell semantics plus Haskell syntax, then Liskell consists of Haskell semantics plus Lisp syntax. Liskell is Haskell on the inside but looks like Lisp on the outside, as in its source code it uses the typical Lisp syntax forms, namely symbol expressions, that are distinguished by their fully parenthesized prefix notation form. Liskell captures the most Haskell syntax forms in this prefix notation form, for instance: `if x then y else z` becomes `(if x y z)`, while `a + b` becomes `(+ a b)`.

Except for aesthetics, there is another argument for Lisp syntax: meta-programming becomes easy. Liskell features a different meta-programming facility than the one found in Haskell with Template Haskell. Before turning the stream of lexed tokens into an abstract Haskell syntax tree, Liskell adds an intermediate processing data structure: the parse tree. The parse tree is essentially is a string tree capturing the nesting of lists with their enclosed symbols stored as the string leaves. The programmer can implement arbitrary code expansion and transformation strategies before the parse tree is seen by the compilation stage.

After the meta-programming stage, Liskell turns the parse tree into a Haskell syntax tree before it sent to the compilation stage. Thereafter the compiler treats it as regular Haskell code and produces a Haskell calling convention compatible output. You can use Haskell libraries from Liskell code and vice versa.

Liskell is implemented as an extension to GHC and its darcs branch is freely available from the project's website. The Liskell Prelude features a set of these parse tree transformations that enables traditional Lisp-styled meta-programming as with defmacro and backquoting. The project's website demonstrates meta-programming application such as proof-of-concept versions of embedding Prolog inference, a minimalistic Scheme compiler and type-inference in meta-programming.

The future development roadmap includes stabilization of its design, improving the user experience for daily programming – especially error reporting – and improving interaction with Emacs.

Further reading

http://liskell.org

3.1.2 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 has a running implementation for small tests but, like most ports of languages to Palm OS, we are dealing with memory allocation issues. Also, other higher priority projects have now intervened so this project is going into the background for a while.

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.

3.2 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 (→ 3.2.2) 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) (→ 7.3.9) 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.

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 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, Mischa Dieterle

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 6.7 is available on request. 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

- Steffen Priebe: *Structured Generic Programming in Eden*, Department of Mathematics and Computer Science Philipps-Universitaet Marburg, February 2007.
- Jost Berthold and Rita Loogen: *Visualising Parallel Functional Program Runs - Case Studies with the Eden Trace Viewer*, Parallel Computing (ParCo) 2007, September 2007.
- Jost Berthold, Mischa Dieterle, Rita Loogen, Steffen Priebe: *Hierarchical Master-Worker Skeletons*, Symposium on Trends in Functional Programming (TFP), New York, April 2007.
- Jost Berthold, Abyd Al-Zain, and Hans-Wolfgang Loidl: *Adaptive High-Level Scheduling in a Generic Parallel Runtime Environment*, Symposium on Trends in Functional Programming (TFP), New York, April 2007.
- Jost Berthold, Rita Loogen: *Parallel Coordination Made Explicit in a Functional Setting.* In Zoltón Horáth and Viktória Zsók, editors, 18th Intl. Symposium on the Implementation of Functional Languages (IFL 2006), LNCS 4449, pp 73–90, Springer 2007. Awarded best paper of IFL 2006 (Peter Landin-Prize 2006).
- Mercedes Hidalgo-Herrero, Yolanda Ortega-Mallén, Fernando Rubio: *Comparing Alternative Evaluation Strategies for Stream-based Parallel Functional Languages.* In Zoltón Horáth and Viktória Zsók, editors, 18th Intl. Symposium on the Implementation of Functional Languages (IFL 2006), LNCS 4449, Springer 2007.
- 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, Electronic Notes in Theoretical Computer Science, to appear.

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 (→ 4.8.3). 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 extensional 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.

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

Latest developments

The latest developments mostly concern the transfer of ideas/methods found in Chameleon to other systems. For example, implication constraints as pioneered in Chameleon have found their way into GHC 6.6. We also plan to integrate some of Chameleon's type inference capabilities into Tim Sheard's Omega.

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: http://taichi.ddns.comp.nus.edu.sg/taichiwiki/XhaskellHomePage

Latest developments

We are currently working on the integration of type classes. A new version is planned for June 2007.

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

Latest developments

We are working on a much improved version which is scheduled for June 2007.

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) (→4.3.3) 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 (→5.5.3) (included in the EHC project).

The EHC project also tackles other issues:

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

- Each step corresponds to an actual, that is, an executable compiler. Each of these compilers is a compiler in its own right so experimenting can be done in isolation of additional complexity introduced in later steps.
- The description of the compiler uses code fragments which are retrieved from the source code of the compilers. In this way the description and source code are kept synchronized.

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

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

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

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 (→5.5.3).

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

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

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

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

Although our type system is developed in the context of the language Clean, it is also relevant to Haskell because the core uniqueness type system we propose is very similar to the Haskell's core type system. Moreover, we are currently working on defining syntactic conventions, which programmers can use to write type annotations, and compilers can use to report types, without mentioning uniqueness at all.

Further reading

- Edsko de Vries, Rinus Plasmeijer and David Abrahamson, "Equality-Based Uniqueness Typing". Presented at TFP 2007, submitted for post-proceedings.
- Edsko de Vries, Rinus Plasmeijer and David Abrahamson, "Uniqueness Typing Redefined", in *Z. Horváth, V. Zsók, and Andrew Butterfield (Eds.): IFL 2006, LNCS 4449* (to appear).

3.3.7 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

3.3.8 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.

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 Test.IOSpec

Report by:	Wouter Swierstra
Status:	active development

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

The Test.IOSpec library is essentially a drop-in replacement for several other modules, most notably Data.IORef and Control.Concurrent. Once you're satisfied that your functions are reasonably well-behaved with respect to the pure specification, you can drop the Test.IOSpec import in favour of the "real" IO modules.

There's still quite some work to be done. First and foremost, I'd like to make it easier to combine different modules. Furthermore, I'd also like to add new modules providing specifications of other parts of the IO monad: Control.Concurrent.STM and Control.Exception are two prime candidates.

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

Further reading

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

4.2.2 PFP – Probabilistic Functional Programming Library for Haskell

Report by:	Martin Erwig
Participants:	Steve Kollmansberger
Status:	mostly stable

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. For the next version, some refactorings are planned. Several variations of this library seem to have evolved. Somebody is also working on a documentation. Maybe all the different threads should be brought together on one web page?

Further reading

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

4.2.3 GSLHaskell

Report by:	Alberto Ruiz
Status:	active development

GSLHaskell is a simple library for linear algebra and numerical computation, internally implemented using GSL, BLAS and LAPACK. The goal is to achieve the functionality and performance of GNU-Octave and similar systems. Recent dev elopments include important bugfixes and the interface to additional LAPACK functions. A brief manual is available at the URL below.

This library is used in the *easyVision* project (→ 6.19).

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 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.

The library should still be considered alpha quality. A backend for sparse vector types is near completion, which allows low-overhead “views” of tensors as arbitrarily nested vectors. For instance, a matrix, which we represent as a tuple-indexed vector, could also be seen as a (rank 1) vector of (rank 1) vectors. These different views usually produce different behaviours under common vector operations, thus increasing the expressive power of the interface.

Further reading

- Original announcement:
  http://article.gmane.org/gmane.comp.lang.haskell.general/13561
- Library:
  http://ofb.net/~frederik/stla/

4.2.5 Haskell Rules: Embedding Rule Systems in Haskell

Report by:	Martin Erwig
Participants:	Steve Kollmansberger
Status:	mostly stable

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.

No changes since last report. No plans for future versions.

Further reading

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

4.3 Parsing and transforming

4.3.1 InterpreterLib

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

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

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

Further reading

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

Contact

⟨nfrisby@ittc.ku.edu⟩

4.3.2 hscolour

Report by:	Malcolm Wallace
Status:	stable, maintained

HsColour is a small command-line tool (and Haskell library) that syntax-colorises Haskell source code for multiple output formats. It consists of a token lexer, classification engine, and multiple separate pretty-printers for the different formats. Current supported output formats are ANSI terminal codes, HTML (with or without CSS), and LaTeX. In all cases, the colours and highlight styles (bold, underline, etc) are configurable. It can additionally place HTML anchors in front of declarations, to be used as the target of links you generate in Haddock documentation.

HsColour is widely used to make source code in blog entries look more pretty, to generate library documentation on the web, and to improve the readability of ghc's intermediate-code debugging output.

Further reading

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

4.3.3 Utrecht Parsing Library and Attribute Grammar System

Report by:	Doaitse Swierstra and Jeroen Fokker
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.

  Since the last report several improvements were made: better error reporting of cyclic dependencies, and a large speed improvements in the overall flow analysis have been made. The first versions of the EHC now compile without circularities, nor direct nor induced by fixing the attribute evaluation orders

- we are adding better support for higher order attribute grammars and forwarding rules
- Tthe error correcting strategies of the parser combinators are now being used as a base for providing automatic feedback in systems for training strategies (Johan Jeuring, Arthur van Leeuwen)
- a start has been made with providing Haddock information with the code of the parser combinators
- we plan to enhance the parser combinators with a second basic parsing engine, in order to support monadic uses of the combinators while keeping the error correcting capabilities

The software is again available through the Haskell Utrecht Tools page. (http://www.cs.uu.nl/wiki/HUT/WebHome).

4.3.4 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

4.3.5 RecLib – A Recursion and Traversal Library for Haskell

Report by:	Martin Erwig
Participants:	Deling Ren
Status:	mostly stable

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 inntermost 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.

No changes since last report. No plans for future versions.

Further reading

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

4.4 System

4.4.1 Harpy

Report by:	Martin Grabmüller and Dirk Kleeblatt
Status:	experimental

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

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

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

We made an initial public release, and are now looking forward to ideas from the community to show some further uses of run-time code generation in Haskell.

Further reading

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

4.4.2 hs-plugins

Report by:	Don Stewart
Status:	maintained

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

hs-plugins has been ported to GHC 6.6.

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/

4.4.3 The libpcap Binding

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

Nicholas Burlett has now created a cabalized version and made it available on hackage. However, beware that this doesn't use autoconf to check your system supports `sa_len` and it doesn't check which version of libpcap is installed. It will probably work but may not. If it doesn't then try this:

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

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

All contributions are welcome especially if you know how to get cabal to run autoconf and check for versions of non-Haskell libraries.

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 (→ 4.7.1))
- 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 library has now been incorporated into the set of standard libraries distributed with all Haskell compilers. From GHC 6.6.1 onwards, the filepath library is always available. Version 1.0 of this library has been released, and no further changes are envisaged.

Further reading

http://www-users.cs.york.ac.uk/~ndm/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 and Oleg Kiselyov
Status:	active development

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

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

Currently we fully support Oracle, Sqlite, and PostgreSQL.

Since the last report we have:

- added support for re-usable connections, which should help applications that want to implement a connection pool

- improved the installation process, and added a README file

- replaced the buggy UTF8 module with a correct, tested module

- support for Oracle output bind variables and cursor result-sets

- various bug-fixes, including: Postgres execDML failures, Postgres bound statement failures, Sqlite execDML rowcount incorrect.

Future plans

- ODBC interface. Work has started on this.
- MS SQL Server interface, via COM.

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 Data.ByteString

Report by:	Don Stewart
Status:	active development

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

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

Work on Data.ByteString continues. In particular, a new fusion mechanism, stream fusion, has been developed, which should further improve performance of ByteStrings. This work is described in the recent "Stream Fusion: From Lists to Streams to Nothing at All" paper. Data.ByteString has recently been ported to nhc98.

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

4.6.3 Data.List.Stream

Report by:	Don Stewart
Status:	active development

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

Further reading

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

4.6.4 dimensional

Report by:	Bjorn Buckwalter
Status:	active, unstable

Dimensional is a library providing data types for performing arithmetic with physical quantities and units. Information about the physical dimensions of the quantities/units is embedded in their types and the validity of operations is verified by the type checker at compile time. The boxing and unboxing of numerical values as quantities is done by multiplication and division with units. The library is designed to, as far as is practical, enforce/encourage best practices of unit usage.

Dimensional is currently in a pre-1.0 state and is being actively developed. The most recent release can be downloaded from the project web site (follow url below). Immediate plans include adding more units, tightening exports and getting the library in shape for a 1.0 release (patches are welcome). The library name and module hierarchy are likely to change before the 1.0 release. For more details see the “Issues” section of the project web site.

Further reading

http://code.google.com/p/dimensional/

4.6.5 Numeric prelude

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

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

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

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

There is also a collection of mathematical object types, which is useful both for applications and testing the class hierarchy. The types are
- complex number, quaternion,
- residue class,
- fraction,
- partial fraction,
- numbers equipped with physical units (dynamic checks only),
- fixed point arithmetic with respect to arbitrary bases and numbers of fraction digits,
- infinite precision number in an arbitrary positional system as lazy lists of digits supporting also numbers with terminating representations,
- polynomial, power series, Laurent series
- root set of a polynomial,
- matrix (basics only),
- algebra, e.g. multi-variate polynomial (basics only),
- permutation group.

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

Using the revised system requires hiding some of the standard functions provided by Prelude, which is fortunately supported by GHC (→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 polymorphic 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. The dual application is extensible polymorphic variants (open unions). HList contains several implementations of open records, including records as sequences of field values, where the type of each field is annotated with its phantom label. We and now others (Alexandra Silva, Joost Visser: PURe.CoddFish project (→ 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.

The HList repository is available via Darcs (→ 6.10): http://darcs.haskell.org/HList

Einar Karttunen has made adjustments to HList for GHC 6.6 and Cabalized it.

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 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⟩

4.7.2 binary

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

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

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

The binary library has been heavily tuned for performance, particularly for writing speed. Throughput of up to 160M/s has been achieved in practice, and in general speed is on par or better than NewBinary, with the advantage of a pure interface. Efforts are underway to improve performance still further. Plans are also taking shape for a parser combinator library on top of binary, for bit parsing and foreign structure parsing (e.g. network protocols).

Data.Derive (→ 5.3.1) has support for automatically generating Binary instances, allowing to read and write your data structures with little fuzz.

Binary was developed by a team of 8 during the Haskell Hackathon, and since then has in total 15 people contributed code and many more given feedback and cheerleading on `#haskell`.

The underlying code is currently being rewritten to give even better performance – both reading and writing – still exposing the same API.

The package is is available through Hackage.

Further reading

- Homepage
  http://www.cse.unsw.edu.au/~dons/binary.html
- Hackage
  http://hackage.haskell.org/cgi-bin/hackage-scripts/package/binary
- Development version
  `darcs get -partial`
  http://darcs.haskell.org/binary

4.7.3 binarydefer

Report by:	Neil Mitchell

The Binary Defer library provides a framework for doing binary serialisation, with support for deferred loading. Deferred loading is for when a large data structure exists, but typically only a small fraction of this data structure will be required. By using deferred loading, some of the data structure can be read quickly, and the rest can be read on demand, in a pure manner.

This library is at the heart of Hoogle 4 (→ 5.5.7), but has already found uses outside that application, including to do offline sorts etc.

Further reading

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

4.7.4 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⟩

4.7.5 The Haskell Cryptographic Library

Report by:	Dominic Steinitz

Following the Haskell hackathon http://haskell.org/haskellwiki/Hac_2007, there is now a new release of the library, 4.0.3, which contains only the cryptographic functions and not the functions to handle ASN.1, X.509, PKCS#8 and PKCS#1.5.

This means no dependency on NewBinary which had been requested by several people.

Very limited details of an embryonic version of the ASN.1 library (4.7.6) are available from http://haskell.org/asn1.

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

I have restructured SHA-1 to be more Haskell-like and it's now obvious how it mirrors the specification. However, this has led to rather poor performance and it's not obvious (to me at least) what can be done without sacrificing clarity.

This release contains:
- DES
- Blowfish
- AES
- Cipher Block Chaining (CBC)
- PKCS#5 and nulls padding
- SHA-1
- MD5
- RSA
- OAEP-based encryption (Bellare-Rogaway)

Further reading

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

4.7.6 The Haskell ASN.1 Library

Report by:	Dominic Steinitz

Following the Haskell hackathon http://haskell.org/haskellwiki/Hac_2007, there is now a release of the ASN.1 library, 0.0.1, which contains functions to handle ASN.1, X.509, PKCS#8 and PKCS#1.5.

This still has a dependency on NewBinary.

The current version handles the Basic Encoding Rules (BER) only. I am currently thinking about how to make it handle the Packed Encoding Rules. This will require handling subtype constraints which are currently not supported (they are ignored in BER).

This release supports:
- X.509 identity certificates
- X.509 attribute certificates
- PKCS#8 private keys
- PKCS#1 version 1.5

Further reading

http://haskell.org/asn1.

4.7.7 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 Grapefruit – A declarative GUI library

Report by:	Wolfgang Jeltsch
Participants:	Wolfgang Jeltsch, Matthias Reisner, Daniel Skoraszewsky
Status:	provisional

Grapefruit is a library for creating graphical user interfaces in a declarative way. It is currently based on Gtk2Hs (→ 4.8.3) but implementations on top of other GUI libraries are planned for the future.

Grapefruit makes it possible to implement graphical user interfaces by describing them as systems of interconnected components. Components can be visible components like widgets and windows but also invisible components which provide certain control functionality. Component systems can be build from components by using methods from the `Arrow` and `ArrowLoop` classes.

Components communicate via signals and event streams. A signal denotes a time-dependent value, and an event stream denotes a sequence of events occurring at discrete points in time. Several functions allow the construction of signals and event streams in a purely functional manner.

With Grapefruit, user interface descriptions always cover the complete lifetime of the respective interface. No explicit event handler registrations and no explicit recalculations of values are necessary. This is in line with the declarative nature of Haskell because it stresses how the user interface operates instead of how this operation is achieved. Internally though, signals and event streams are implemented efficiently using the event dispatching and handling mechanism of the underlying GUI toolkit.

The roots of Grapefruit lie in systems like FranTk and wxFruit. Grapefruit tries to combine concepts of these systems with new ideas to become a system which maintains a reasonable balance between ease of use and efficiency.

As of May 2007, Grapefruit is in an early stage. The basic concepts are implemented but the implementation is still subject to notable change. Features planned for the next months cover support for dynamic user interfaces and animations as well as better handling of simultaneous event occurrences.

Grapefruit's source code and documentation can be accessed via its wiki page (see below).

Further reading

http://haskell.org/haskellwiki/Grapefruit

4.8.2 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

4.8.3 Gtk2Hs

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

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

GUIs written using Gtk2Hs 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
- high quality vector graphics using Cairo
- extensive reference documentation
- an implementation of the “Haskell School of Expression” graphics API
- support for the Glade visual GUI builder
- bindings to some Gnome extensions: GConf, a source code editor widget and a widget that embeds the Mozilla/Firefox rendering engine
- an easy-to-use installer for Windows
- packages for Fedora, Gentoo (→ 7.4.2), Debian and FreeBSD

The Gtk2Hs library is actively maintained and developed. We had a major new release back in February. We expect to do another minor release shortly which will contain various bug fixes and minor additions. This will include drag and drop support, c2hs fixes and support for Postscript and PDF output from the Cairo vector graphics library.

In the medium term we hope to support the new features in Gtk+ 2.10, to improve the signals API. In the longer term we hope to modularise Gtk2Hs and enable it to be built and distributed with Cabal and Hackage.

We are always keen to get more people involved with Gtk2Hs. There are plenty of potential coding projects and we are looking for help with writing tutorials. We are interested in feedback from people using Gtk2Hs and especially in interesting applications that we can show off on the website.

Further reading

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

4.8.4 hscurses

Report by:	Stefan Wehr
Status:	stable/beta

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

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

The binding itself is stable; however, the widget library is still beta. 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/

4.8.5 VTY

Report by:	Stefan O'Rear

VTY (Virtualized tTY) is a terminal control library, similar to Stefan Wehr’s hscurses (→ 4.8.4). However vty is designed to have a much easier to use API; all communication is accomplished using 5 functions (most using only 2), with a simple data type. Code which describes screen images is pure and declarative. Vty supports all generally useful features of the Linux terminal emulator except for palette setting. It is used successfully by Shellac, Yi (→ 6.12), and the author's unpublished HsLife program.

Current disadvantages are poor support for non-Linux terminals, poor performance, and a lack of interested hacking/maintainership.

Further reading

- Source repository:
  darcs get http://members.cox.net/stefanor/vty

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 (→ 4.8.2) 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 (→ 7.4.1), 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 (→ 6.16).)

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 (see: Magnus Jonsson: Haskore microtonal support http://www.haskell.org/pipermail/haskell-cafe/2006-September/018144.html).
- 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 tagsoup

Report by:	Neil Mitchell

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

The library provides a basic data type for a list of unstructured tags, a parser to convert HTML into this tag type, and useful functions and combinators for finding and extracting information. The library has seen real use in an application to give Hackage listings, and is used in the next version of Hoogle (→ 5.5.7).

Further reading

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

4.10.5 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 for ghc-6.6.x.

The development version (currently at 1.18, also available through a darcs repository) includes a much-requested lazy parser, and a SAX-like streaming parser. Only some minor work still remains, to tidy things up before the development version is tagged and released as stable.

We recently split off the new lazy parser combinators used by HaXml into a separate library package called polyparse.

Further reading

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

4.10.6 Haskell XML Toolbox

Report by:	Uwe Schmidt
Status:	sixth major release (current release: 7.1)

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 a validator based on DTDs and a new more powerful validator for Relax NG schemas.

The Haskell XML Toolbox bases on the ideas of HaXml (→ 4.10.5) 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
- darcs repository with current development version (7.2) under http://darcs.fh-wedel.de/hxt

Current Work

A master thesis has been finished developing an XSLT system. The result is a rather complete implementation of an XSLT transformer system. Only minor features are missing. The implementation consists of about only 2000 lines of Haskell code. The XSLT module is included since the HXT 7.0 release.

A second master student's project, the development of a web server called Janus, has been finished in October 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 processing all internal data of this web server. The Janus server is highly configurable and can be used not only as HTTP server, but for various other server like tasks. The results of this work will be available via a darcs repository in June 2007. Current activity consists of testing, example applications, demos and documentation.

A new project, an application for HXT and Janus will start in summer 2007: Two master students will construct an index and search engine for specialized search tasks. This system will be highly configurable, such that tasks like searching within a web site, search of articles within a book store or search within a newspaper archive becomes possible. Distribution of the index and search engines within a network architecture will be an additional aspect of this project.

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, the arrow based Relax NG validator and the XSLT system. A getting started tutorial about HXT is avaliable in the Haskell Wiki (http://www.haskell.org/haskellwiki/HXT).

4.10.7 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 (→ 4.4.2)) as well as the implementation of a session as a continually running server thread
- Transactional interface to server-side variables and to databases. The interface is inspired by the work on STM (software transactional memory), but modified to be useful in the context of web applications. The interface relies on John Goerzens hdbc package and its PostgreSQL driver.

Current work includes

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

Further reading

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

5 Tools

5.1 Foreign Function Interfacing

5.1.1 C→Haskell

Report by:	Manuel Chakravarty
Status:	active

C→Haskell is an interface generator that simplifies the development of Haskell bindings to C libraries. 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.

Duncan Coutts has been busy implementing a new C parser that is very closely aligned to gcc's grammar and has been tested on a large pool of open source code. We are planning a new release including the new parser and many bug fixes soon; in the meantime, all of the new code is available from the darcs repository. More information is at http://www.cse.unsw.edu.au/~chak/haskell/c2hs/.

5.2 Scanning, Parsing, Analysis

5.2.1 Alex version 2

Report by:	Simon Marlow
Status:	stable, maintained

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

The latest release is version 2.1.0.

Changes in version 2.1.0:

- Alex is now in a Darcs repository (→ 6.10), 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/

5.2.2 Happy

Report by:	Simon Marlow
Status:	stable, maintained

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

The latest release is 1.16, released 8 January 2007. There have been no changes to the darcs sources since 1.16, but I have some pending changes to fix one annoying bug (Happy crashes instead of emitting error messages), and I have some changes that speed up Happy by 10% or so.

Further reading

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

5.2.3 SdfMetz

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

SdfMetz supports grammar engineering by calculating grammar metrics and other analyses. Currently it supports four different grammar formalisms (SDF, DMS, Antlr and Bison) from which it calculates size, complexity, structural, and ambiguity metrics. Output is a textual report or in Comma Separated Value format. The additional analyses implemented are 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*.

Future plans

A web-based interface is planned and more metrics will be added. 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.

5.2.4 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 derive

Report by:	Neil Mitchell and Stefan O'Rear
Participants:	Neil Mitchell, Stefan O'Rear, Twan van Laarhoven, Spencer Janssen, Andrea Vezzosi

The instance derivation mechanism in Haskell is useful, but it has too little power for many uses. User defined classes cannot be derived automatically even when there is an obvious algorithm to do it. Previous attempts, such as DrIFT, are restricted in output form and closed in the supported classes.

Data.Derive is designed to rectify both of these issues. By design, implementing derivation for a new class is extremely simple: add a single module to the GHC search path. Data.Derive uses an Abstract Syntax Tree for output, allowing us to operate both as a preprocessor a la DrIFT and Template Haskell based build integration.

The Derive tool has also attracted new features not present in DrIFT. Twan van Laarhoven has implemented deriving support for Functor, as proposed for Haskell'. Neil Mitchell has done some work on guessing inductive instances, allowing users to specify an example of the instance, and then automatically inferring the rules to derive it.

Further reading

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

5.3.2 Term Rewriting Tools written in Haskell

Report by:	Salvador Lucas

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

The tools have been developed at the Technical University of Valencia (UPV) as part of a number of research projects. The following people is (or has been) involved in the development of these tools: Beatriz Alarcón, María Alpuente, Demis Ballis (Università di Udine), Santiago Escobar, Moreno Falaschi (Università di Siena), Javier García-Vivó, Raúl Gutiérrez, José Iborra, Salvador Lucas, Rafael Navarro, 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 (→ 4.8.2).

Immediate plans

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

References

- Building .NET GUIs for Haskell applications. B. Alarcón and S. Lucas. 6th International Conference on .NET Technologies, pages 57–66, 2006.

- Proving Termination of Context-Sensitive Rewriting With MU-TERM B. Alarcón, R. Gutiérrez, J. Iborra, and S. Lucas. Electronic Notes in Theoretical Computer Science, to appear, 2007.

- Abstract Diagnosis of Functional Programs M. Alpuente, M. Comini, S. Escobar, M. Falaschi, and S. Lucas Selected papers of the International Workshop on Logic Based Program Development and Transformation, LOPSTR'02, LNCS 2664:1–16, Springer-Verlag, Berlin, 2003.

- OnDemandOBJ: A Laboratory for Strategy Annotations M. Alpuente, S. Escobar, and S. Lucas 4th International Workshop on Rule-based Programming, RULE'03, Electronic Notes in Theoretical Computer Science, volume 86.2, Elsevier, 2003.

- Connecting Remote Tools: Do it by yourSELF! M. Alpuente and S. Lucas. ERCIM News 61:48–49, April 2005.

- MU-TERM: A Tool for Proving Termination of Context-Sensitive Rewriting S. Lucas 15th International Conference on Rewriting Techniques and Applications, RTA'04, LNCS 3091:200–209, Springer-Verlag, Berlin, 2004.

- A Rule-based System for Web site Verification. Demis Ballis and Javier García-Vivó. 1st International Workshop on Automated Specification and Verification of Web Sites, WWV'05, Valencia (SPAIN). Electronic Notes in Theoretical Computer Science, 157(2):11–17, 2006.

5.3.3 HaRe – The Haskell Refactorer

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

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

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

In order to allow users to extend HaRe themselves, HaRe includes an API for users to define their own program transformations, together with Haddock (→ 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/

5.3.4 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 obtain 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 (→ 4.7.7).

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

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 GHC 6.8.

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.

Since the last HCAR report, there have been two significant developments in Hpc camp.

- An Ajax based tracer has been developed that uses the Hpc ticks to highlight actual control flow inside a Haskell program using a browser view of Haskell source code. Unsurprisingly lazy functional code jumps around in a semi-understandable manner. The tracer turns out to be useful for finding errors like head of `[]`, because the tracer can run till the exception is raised, then replay the control flow backwards, showing what code fragment causes the bad call to head.

- Hpc now has a small DSL for specifying code fragments that should be ignored when computing and displaying coverage. This DSL can be used to help classify things like code that is genuinely expected to never be called, for example if the code can only be reached when a higher-level precondition has been violated. The DSL can also be used to tag test code to be ignored when considering system level coverage. Another use case is capturing the ignoring of idioms that are expected to contain non-executed code. This DSL is provided as a processor for the open Hpc file formats, and works with the other Hpc tools.

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:	maintenance

The Haskell tracing system Hat is based on the idea that a specially compiled Haskell program generates a trace file alongside its computation. This trace can be viewed in various ways with several tools: 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.

During the last half year only small bug fixes were committed to the darcs (→ 6.10) repository. But as part of the Google Summer of Code, Kenn Knowles has been funded to update Hat to work well with the large number of Haskell libraries used by almost all real coding projects these days. Hopefully this will resurrect Hat to a state where it can be used once again by ordinary people on a regular basis to help them understand the behaviour of their code.

Several other updates are also planned for the near future, including new and improved trace-browsers.

Further reading

- A Theory of Tracing Pure Functional Programs http://www.cs.kent.ac.uk/people/staff/oc/traceTheory.html
- http://www.haskell.org/hat
- darcs get http://darcs.haskell.org/hat
- Google Summer of Code project http://code.google.com/soc/haskell/appinfo.html?csaid=637BFC2B6B13D512

5.4.3 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 (→ 7.1.3). 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.

5.5 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 Haskell Modes for Vim

Report by:	Claus Reinke
Participants:	All Haskell & Vim users
Status:	ongoing

While there is no single best Haskell mode for Vim, there are certainly numerous Vimmers out there with their own personalized Haskell mode settings for Vim, and there are certainly numerous Haskellers out there looking for the kind of IDE functionality that some Vimmers have at their fingertips already.

Since surprisingly many Haskellers are not quite aware of Vim's IDE functions, I have created a little introductory (and incomplete) tour of screenshots (for more information, see Vim's excellent built-in :help, or browse the help files online at http://vimdoc.sourceforge.net/htmldoc/usr_toc.html). I have also replaced my own old Hugs-based Vim scripts, which have been online for years, with my current, GHC- and Haddock-based scripts. The tour gives an overview of what functionality they provide.

1. There is a section at haskell.org listing several people's Vim files:

   http://www.haskell.org/haskellwiki/Libraries_and_tools/Program_development#Vim

2. My own Vim scripts and plugins are available here (just updated):

   http://www.cs.kent.ac.uk/people/staff/cr3/toolbox/haskell/Vim/

3. A short tour of some Vim support for Haskell editing (screenshots):

   http://www.cs.kent.ac.uk/people/staff/cr3/toolbox/haskell/Vim/vim.html

I hope that (1) might encourage more Haskell Vimmers to link to their own tricks and tips (perhaps there should be a top-level 'Haskell modes for Vim' section at haskell.org, similar to the 'Haskell mode for Emacs' section that is already there), (2) might be useful to some of you, and (3) might help to motivate some of you to give Vim a try. It is really not as if Vim (or Emacs, for that matter) didn't have more IDE functionality than most of us ever use, it is more that there is so much of it to learn and to fine-tune to your personal preferences.

5.5.3 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.

5.5.4 cpphs

Report by:	Malcolm Wallace
Status:	stable, maintained

Cpphs is a robust drop-in Haskell replacement for the C pre-processor. It has a couple of benefits over the traditional cpp – you can run it in Hugs when no C compiler is available (e.g. on Windows); and it understands the lexical syntax of Haskell, so you 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 1.4, containing some minor bugfixes, especially to macro expansions in cpp conditionals.

Further reading

http://haskell.org/cpphs

5.5.5 Visual Haskell

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

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

Version 0.2 of Visual Haskell was released in December 2006. It includes support for Visual Studio 2005, and comes with GHC 6.6.

The sources are in a darcs (→ 6.10) repository here: http://darcs.haskell.org/vshaskell/, and are provided with a BSD-license. 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!

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

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.

Work continues on a new version of Haddock based on the GHC API; this will become version 2.0.

Changes since the 0.8 release:

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

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. Some libraries required for Hoogle 4 have been developed and released, including tagsoup (→ 4.10.4) and binarydefer (→ 4.7.3).

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

Report by:	Don Stewart
Status:	active development

Xmonad is a minimalist tiling window manager for X, written in Haskell. Windows are managed using automatic layout algorithms, which can be dynamically reconfigured. At any time windows are arranged so as to maximise the use of screen real estate. All features of the window manager are accessible purely from the keyboard: a mouse is entirely optional. Xmonad is configured in Haskell, and custom layout algorithms may be implemented by the user in config files. A principle of Xmonad is predictability: the user should know in advance precisely the window arrangement that will result from any action.

By default xmonad provides three layout algorithms: tall, wide and fullscreen. In tall or wide mode, windows are tiled and arranged to prevent overlap and maximise screen use. Sets of windows are grouped together on virtual screens, and each screen retains its own layout, which may be reconfigured dynamically. Multiple physical monitors are supported via Xinerama, allowing simultaneous display of a number of screens.

Further reading

- Home page:
  http://xmonad.org/
- Darcs source:
  `darcs get` http://darcs.haskell.org/~sjanssen/xmonad
- IRC channel:
  `#xmonad @ irc.freenode.org`
- Mailing list:
  ⟨xmonad@haskell.org⟩

6.2 GenI

Report by:	Eric Kow
Status:	active development

GenI is a surface realiser for Tree Adjoining Grammars. Surface realisation can be seen as the last stage in a natural language generation pipeline. GenI in particular takes an FB-LTAG grammar and an input semantics (a conjunction of first order terms), and produces the set of sentences associated to the input semantics by the grammar. It features a surface realisation library, several optimisations, batch generation mode and a graphical debugger written in wxHaskell. It was developed within the TALARIS project and is free software licensed under the GNU GPL.

Further reading

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

6.3 Roguestar

Report by:	Christopher Lane Hinson
Status:	early development

Roguestar is a science fiction role playing game. In gameplay, it belongs to the roguelike family of games. It sports an OpenGL interface. Roguestar is licensed under the GNU GPL.

Roguestar is in the early stages of development. It is not yet "fun."

I am currently refactoring the roguestar graphics and animation library, which will be available under a permissive license. The next goal is to implement combat and a simple AI.

Further reading

http://roguestar.downstairspeople.org

6.4 mmisar

Report by:	Slawomir Kolodynski
Status:	under development

mmisar is a tool supporting translation of formalized mathematics from the Metamath's set.mm to the Isar formal proof language so that it can be verified by Isabelle/ZF. I created it for my IsarMathLib project (a library of formalized mathematics for Isabelle/ZF). As of release 1.4.0 IsarMathLib contains about 1000 facts and 500 proofs translated from Metamath to Isabelle/ZF with mmisar. The tool is included in the distribution of the IsarMathLib project and licensed under GPL. It is under active development as I am using it to learn Haskell. In the next release I am planning to rewrite the parser for Metamath ZF formulas to base it on Parsec.

Further reading

- http://savannah.nongnu.org/projects/isarmathlib
- http://us.metamath.org/
- http://www.cl.cam.ac.uk/research/hvg/Isabelle/

6.5 Inference Services for Hybrid Logics

Report by:	Carlos Areces, Daniel Gorin, Guillaume Hoffmann

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

The Talaris group at Loria, Nancy, France (http://talaris.loria.fr) and the GLyC group at the Computer Science Department of the University of Buenos Aires, Argentina (http://www.glyc.dc.uba.ar/) are developing a suite of tools for automated reasoning for hybrid logics. Most of them are (successfully) written in Haskell. A brief description of some of these tools follows.

6.5.1 HyLoRes

Report by:	Carlos Areces, Daniel Gorin, Guillaume Hoffmann
Status:	active development
Current release:	2.1

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

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

Further reading

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

6.5.2 HTab

Report by:	Carlos Areces, Daniel Gorin, Guillaume Hoffmann
Status:	active delopment
Current release:	1.0

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

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

Further reading

- Guillaume Hoffmann. *Terminating tableau algorithms for hybrid logic*. Master Thesis.
- Source:
  http://trac.loria.fr/projects/htab

6.5.3 HGen

Report by:	Carlos Areces, Daniel Gorin, Guillaume Hoffmann
Status:	active development
Current release:	1.0

HGen is a random CNF (conjunctive normal form) generator of formulas for different hybrid logics. It is highly parametrized to obtain tests of different complexity for the different languages. It has been extensively used in the development of HyLoRes (→6.5.1) and HTab (→6.5.2).

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

Further reading

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

6.6 Raskell

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

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

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

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

Further reading

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

Contact

⟨alex@ittc.ku.edu⟩

6.7 photoname

Report by:	Dino Morelli
Status:	stable, maintained

photoname is a command-line utility for renaming/moving photo image files. The new folder location and naming are determined by the EXIF photo shoot date and the usually-camera-assigned serial number, often appearing in the filename.

Further reading

- Project page:
  http://ui3.info/d/proj/photoname.html
- Source repository:
  darcs get http://ui3.info/darcs/photoname

6.8 HJS – Haskell Javascript Interpreter

Report by:	Mark Wassell
Status:	in development

HJS is a Javascript interpreter and is based on the grammar and behaviour as specified in ECMA-262, 3rd Edition, with additions and modifications from JavaScript 1.5. Current status is that all of the language can be parsed and work is underway to complete the core behaviour and the built-in objects and their methods. Possible options for future directions include a pretty printer and providing multiple hosting environments – DOM, WScript and Gtk2Hs are examples.

Further reading

http://www.haskell.org/haskellwiki/Libraries_and_tools/HJS

6.9 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 (→ 4.7.4).

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⟩

6.10 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. Recently, David has been working on a new "hashed inventory" format, a basic building block for an eventual solution to conflicts problem. It has been working well enough now that we are starting to consider switching our repository over in the near future. Also, developer Jason Dagit has been accepted to work on the conflicts problem as part of a Google Summer of Code project. 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

6.11 lambdabot

Report by:	Don Stewart
Status:	active development

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

New features since the last release include: multi-server mode, configuration scripts, stability improvements, extended defaulting, ghc 6.6 support, and compressed binary checkpointing.

Further reading

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

6.12 yi

Report by:	Don Stewart
Status:	active development

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

Yi activity has increased dramatically in the past few months, as Jean-Philippe Bernardy has taken over active development. In particular, Yi is now based on top of the GHC-api (→ 2.1) library, enabling more interactive and dynamic configuration and extension. Significant architectural changes have occurred, including: Vty frontend (→ 4.8.5) replaces Curses frontend; linewrap support; GTK frontend (→ 4.8.3); dynamic Haskell evaluation (like elisp!); new commands may be dynamically defined; Syntax highlighting in GTK frontend.

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/

6.13 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(→ 4.8.2))

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 (→ 6.10).

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

6.14 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 1.12 (January 2007), which is compatible with GHC 6.6 and has (experimental) 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 Emping

Report by:	Hans van Thiel

I'd like to announce the availability of Emping, a utility that reads a table of nominal data, in a csv format that can be generated from Open Office Calc, derives all shortest rules for a selected attribute, and writes them to a .csv file that can be read by OO Calc. See http://j-van-thiel.speedlinq.nl/emp/empug.html for more and a download.

The connection with Haskell is only that it's written in it.

6.16 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 hardcoded 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. In a recent approach we simulate logic programming by a big lazy cycle of function applications (a tied knot, a fixed point).

- 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. `:-)` We have run certain tests using the FastPackedString and the Binary libraries, which are still not satisfying.

Further reading

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

6.17 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. It has been updated to support the latest ByteString api (→ 4.6.2).

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/

6.18 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

6.19 easyVision

Report by:	Alberto Ruiz
Status:	experimental, active development

The *easyVision* project is a collection of libraries for elementary computer vision and image processing appl ications. We take advantage of Haskell's expressive power without any performance loss, since all heavy numerical c omputations are done by optimized libraries: HOpenGL for 3D graphics and user interface, GSLHaskell (→ 4.2.3) for matrix computations, and an experimental binding to Intel's IPP for fast image processing. We use MPlayer for real time image grabbing and video decoding.

The system exploits higher order functions to create very useful abstractions. For example, we use "camera combinators" to define "virtual cameras" which perform any desired image processing on the infinite image sequences generated by other cameras. We can also define elaborate pattern recognition machines by composition of any desired chain of feature extractors and simple classifiers.

This is a very preliminary version, but some applications are already working. The URL below shows a few screenshots.

Further reading

http://alberrto.googlepages.com/easyvision

6.20 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 Credit Suisse Global Modelling and Analytics Group

Report by:	Ganesh Sittampalam

GMAG, the quantitative modelling group at Credit Suisse, has been using Haskell for various projects since the beginning of 2006, with the twin aims of improving the productivity of modellers and making it easier for other people within the bank to use GMAG models.

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

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

So far, we have:

- Added higher-order functions to Excel, implemented via (Haskell) addin code.
- Built tools to transform spreadsheets into directly executable code.
- Written a "lint" tool to check for common errors in spreadsheets.

Current projects include:

- Further work on tools for checking, manipulating and transforming spreadsheets.
- A domain-specific language embedded in Haskell for implementing reusable components that can be compiled into various target forms.

The addition of higher-order functions to Excel has proved very popular, for example giving modellers the ability to duplicate calculations without having to repeat them over a large area of the spreadsheet (which is inflexible and causes maintenance headaches.)

An increasing number of modellers are being exposed directly to Haskell by using our DSL, and they seem to be picking it up fairly quickly. The reusable nature of components makes it easier to quickly build complete models that can be distributed to end-users.

We are hiring: please see http://tinyurl.com/2lqoq9.

Further reading

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

7.1.2 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://csg.csail.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

7.1.3 Galois, Inc.

Report by: Andy Adams-Moran

Galois is hiring! We need 5 new FP and/or product developers by September! Non-U.S. citizens welcome; we just need to be able to get you a work visa quickly. See http://www.galois.com/join.php for descriptions of the open positions, or just send your resume to ⟨jobs at galois.com⟩.

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

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

Galois continues to grow from strength to strength. As of Spring 2007, Galois is 23 engineers strong, with a support staff of 8. We've been profitable and experienced solid growth each of the last three years, and the future looks good too.

Last year, we had a couple of firsts:

- Our first sabbatical visitor: Colin Runciman visited Galois for just under four months during the summer. It was a fruitful visit for all concerned. Working with various Galois engineers (mainly Mark Tullsen and Andy Gill), Colin's visit saw the development of hpc (the Haskell Program Coverage tool) (→5.4.1), SmallCheck (a QuickCheck variant that tests *all* possible values down to a given depth) (→5.4.3), and an initial version of a line debugger for Haskell (stay tuned!) all stem from Colin's time at Galois.

- Our first intern: Matthieu Boesflug spent a few summer months with us last year working mainly on the multi-level wiki project with Isaac Jones, Sigbjørn Finne, and Dylan McNamee.

This year, we're continuing our sabbatical and internship programs. Graham Hutton will be visiting for 6 weeks towards the end of the summer, and we're looking forward to working with Graham. Perhaps we'll see a coherent and powerful theory for concurrency and exceptions emerge; who knows?

As for interns, this summer we've got two: Darin Morrison and Rebekah Leslie, undergraduate and postgraduate students at Portland State University, respectively. Darin and Rebekah will be working on two of our more foundational projects (one building the case for using Haskell as a *deployed* high assurance development language, and the other exploring how to exploit virtualization for security, in a high assurance manner). We're full up for interns this year, but we hope to be able to offer more internships next year. If you're interested in interning at Galois, write us at ⟨jobs at galois.com⟩, with "Internships 2008" in the Subject: line.

We've also been heavily involved in the effort to define a solid set of extensions to Haskell 98, aka the Haskell' project. Things are coming to head there, and we look forward to making the most of the result.

Content is slowly but surely being built for the Functional Programming Consortium web site, thanks to some of the speakers from last year's well-attended and well-received Commerical Users of Functional Programming workshop. If you're interested in getting involved with the Consortium in any way, contact Andy Adams-Moran (⟨adams-moran at galois.com⟩).

Lastly, we're stepping up our community involvement, so you should see a few more Galwegians at conferences and hopefully some open source releases too.

Further reading

http://www.galois.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.

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

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 (→ 6.18)
- Supporting OO-Design technique for functional programmers (→ 3.3.8)
- 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 (→ 6.20).

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 (→ 1.6.1), 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

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. 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. Recently we proved that our method scales up to extended letrec-calculi with constructors as well as letrec-calculi with a parallel or operator.

Semantics for Haskell extended with direct-call I/O

We introduced the *FUNDIO* calculus which proposes a non-standard way to combine lazy functional languages with I/O using non-deterministic constructs. We defined a contextual equivalence depending on the Input/Output behavior of reduction sequences and we proved correctness of a considerable set of program transformations. In particular we have shown correctness of several optimizations of evaluation, including strictness optimizations.

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 `unsafePerformIO` 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.

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.

Semantics and Transformations for Functional Hardware Descriptions

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

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. The focus of current research are extensions of these calculi with potential applications in 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). We have shown 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
- References to all mentioned research topics are collected on the following webpage
http://www.ki.informatik.uni-frankfurt.de/research/HCAR.html

7.3.4 Formal Methods at Bremen University and DFKI Lab Bremen

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

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

We are using Haskell to develop the Heterogeneous tool set (Hets), which consists of parsers, static analyzers and proof tools for languages from the CASL family, such as CASL itself, HasCASL, CoCASL, CSP-CASL and ModalCASL, and additionally OMDoc and Haskell. HasCASL is a language for specification and development of functional programs; Hets also contains a translation from an executable HasCASL subset to Haskell. There is a prototypical translation of a subset of Haskell to Isabelle HOL and HOLCF.

We use the Glasgow Haskell Compiler (GHC 6.6), exploiting many of its extensions, in particular concurrency, multiparameter type classes, hierarchical name spaces, functional dependencies, existential and dynamic types, and Template Haskell. Further tools actively used are DriFT, Haddock (→ 5.5.6), the combinator library Parsec, HaXml (→ 4.10.5), Programatica, Shellac, HXT and haifa-lite.

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

Further reading

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

7.3.5 Functional Programming at Macquarie University

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

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

- Our alpha version of a port of the yhc (→ 2.4) runtime to Palm OS handhelds is working but going into a hiatus (→ 3.1.2).
- Kate Stefanov's PhD thesis on off-the-shelf compression technology for bytecode-based programs was passed. The main results relevant to Haskell folk are good compression ratios for nhc (→ 2.3) bytecode using a very simple extension of the LZW algorithm.
- Matt Robert's work on the implementation of Jay's pattern calculus continues, with a focus on formal semantics and comparison with related pattern-based calculi.
- We are beginning work on a language processor generation project that will likely use Haskell-based DSLs as the specification notations.

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. Chris Brown continues extending HaRe, the Haskell Refactorer (→ 5.3.3). Nik Sultana is working with Simon on formal verification of Haskell refactorings. 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 (→ 4.8.3) in cooperation with Duncan Coutts, Oxford University. Keith Hanna is continuing work on Vital, a document-centered programming environment for Haskell, and on Pivotal, a GHC-based implementation of a similar environment. Claus Reinke has released his rudimentary Haskell mode for Vim (→ 5.5.2), including Haddock browsing and insert-mode completion of library identifiers. He is currently working on embedding small-step operational semantics in Haskell, used as executable specifications of language design experiments.

Further reading

- FP group:
  http://www.cs.kent.ac.uk/research/groups/tcs/fp/
- Refactoring Functional Programs:
  http://www.cs.kent.ac.uk/projects/refactor-fp/
- Hat:
  http://www.haskell.org/hat/
- Gtk2HS:
  http://www.haskell.org/gtk2hs
- Vital:
  http://www.cs.kent.ac.uk/projects/vital/
- Pivotal:
  http://www.cs.kent.ac.uk/projects/pivotal/
- Vim Haskell mode:
  http://www.cs.kent.ac.uk/people/staff/cr3/toolbox/haskell/Vim/

7.3.7 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` (→ 4.6.2), the hs-plugins (→ 4.4.2) library for dynamically loaded type-safe plugins, the interface generator C→Haskell (→ 5.1.1), and the dynamic editor Yi (→ 6.12). Moreover, we are contributing to widely used Haskell software, such as GHC (→ 2.1), xmonad (→ 6.1), and lambdabot (→ 6.11).

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. Associated data types (aka *data type families*) and System F_C are fully implemented in GHC's development version and we are currently implementing associated type synonyms; see http://haskell.org/haskellwiki/GHC/Indexed_types for details. All of this is planned to be included in the 6.8 release of GHC.

Together with GHC HQ, we are busy with finally bringing nested data parallelism to GHC, with a focus to utilise multi-core CPUs. Parts of our implementation are already ready for experimentation, but are currently only suitable for the adventurous. See http://haskell.org/haskellwiki/GHC/Data_Parallel_Haskell for details.

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

7.3.8 Haskell in Romania

Report by: Dan Popa

This is to report some activities of the Ro/Haskell group. Academic year: 2006–2007.

The Ro/Haskell page was initiated during the autumn of 2006 by Dan Popa (Univ. Bacau,RO) as a supplementary source of information for his students of the Formal Languages Course. Haskell is used in Bacau (State Univ.) to teach language(s) implementation.

January 31,2007. A manual of Haskell in Romanian was published by Dan Popa (editor: Editura EduSoft, Bacau). The readers are guided step by step from the first function in Haskell to an expression evaluator, modular monadic parsing and some words concerning monadic semantic implementation.

February,2007: An other book concerning Haskell was published by M.Gontineac (editor: Editura Alexandru Myller, Iasi). A part of the book is focused on Standard Prelude, carefully compiled and explained, other one on the imperative programming in Haskell using the I/O monad. The mathematic foundation of functional languages are presented in the first chapter.

Both books are presented on the Ro/Haskell webpage: http://www.haskell.org/haskellwiki/Ro/Haskell.

Actually the site of the Ro/Haskell group is visited by students from three faculties, which belongs to two (state) universities, from Bacau and Iasi. It was accessed more than 1000–1250 times and the number is growing.

The University from Cluj is also involved in teaching and research Haskell. Please contact them for details.

Books are donated to libraries in Bacau and Iasi, and presented on editor's website: www.edusoft.ro. (Id=81)

Papers were presented in International Conferences like ICMI 45.

7.3.9 SCIence project

Report by: Kevin Hammond

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

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

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

7.4 User groups

7.4.1 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 (→ 5.2.1), happy (→ 5.2.2), haddock (→ 5.5.6) and darcs (→ 6.10).

Support for the GHC head branch continues.

7.4.2 Haskell in Gentoo Linux

Report by: Andres Löh

Unfortunately, Haskell's lazy nature tends to infect the members of the Gentoo Haskell team on a regular basis, so that we haven't made as much progress during the past six months as we would have liked.

While ghc-6.6 has found its way into the main portage tree, it is still hardmasked. One of the reasons is lack of testing, another is that Duncan Coutts is working on a simplified bootstrapping scheme that will get rid of the "virtual" ghc package that we have been using in the past (but that also has been a source of many troubles).

You can access and test the latest versions of the ebuilds we are working on via our darcs (→ 6.10) 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:

Type improvement constraint, local functional dependencies, and a type-level typecase

The type improvement constraint `TypeCast tau1 tau2`, introduced and implemented in the HList paper, holds if the type `tau1` is equal to `tau2` – or can be equal if some type variables in these types are suitably instantiated. Unlike a similar constraint `TypeEq tau1 tau2 HTrue`, `TypeCast` does instantiate type variables in `tau1` and `tau2`.

The type improvement constraint can express local, per instance rather than class-wide, functional dependencies. The TypeCast constraint is especially useful for type-level type introspection: type-level *type-case*. The web page http://okmij.org/ftp/Haskell/typecast.html shows many applications, for example:

- How to write an instance for not-a-function
- Deepest functor: `fmap` over arbitrarily nested collections
- Deep monadic join
- Monadic `sequence` for HLists

◦ Function types as instances of `Num` and embedding of Forth-like languages into Haskell

◦ Resolving overloading ambiguity and controlling the order of instance selection

◦ Solving the `show . read` problem with local functional dependencies and syntactic hints

HSXML: representing XML-like documents as typed S-expressions/code

HSXML is a library for writing and transforming typed semi-structured data in Haskell – in the form of heterogeneous, arbitrarily nested, *typed* S-expressions, which conform to SXML syntax as data and also represent executable Haskell code. HSXML supports the extensible set of 'tags' and statically enforced content model restrictions. A particular application is writing web pages in Haskell. We obtain HTML, XHTML or other output formats by running the Haskell web page in an appropriate rendering monad.

Literally following the S-expression syntax without resorting to the universal type has not been attained in Haskell before. Here is a sample HSXML:

```
[p "Haskell is a general purpose,"
 [[em [[strong "purely"]] "functional"]]
  "programming language"]
```

one may observe the absence of any commas and other list delimiters. Each element such as 'p' and 'em' has a distinct type and may have an arbitrary number of subelements and strings, as permitted by its content model. HSXML extensively relies on local functional dependencies (see above).

The benefit of representing XML-like documents as a typed data structure/Haskell code is static rejection of bad documents – not only those with undeclared tags but also those where elements appear in wrong contexts (e.g., a 'p' element appearing within `H1`).

The web page http://okmij.org/ftp/Scheme/xml.html#typed-SXML gives further motivation and description. It points to an example of authoring web pages in HSXML and a complex example of context-sensitive HSXLT transformations: producing structurally distinct HTML and XML/RSS from the same master file.

A poly-variadic function of a non-regular type

We describe the implementation of the overloaded function of the type $(\mathrm{Int} \to)^n ([]^n e) \to e \to ([]^n e)$, which replaces an element in a multi-dimensional list. The function is overloaded to handle lists of any dimensions, and so has the variable number of index arguments specifying the location of the element to replace. The number of index arguments, of the type Int, must match the dimensionality of the list.

The gist of the solution, implemented by Chung-chieh Shan, is the case analysis of the type of function's continuation. Or: bottom-up deterministic *parsing of the type of the continuation*. This puzzle demonstrates the benefits of bringing the tools from quite a different area of computer science – parsing and grammars – to seemingly unrelated type class programming. Types and parsing are, of course, deeply related: cf. type logical grammars.

http://okmij.org/ftp/Haskell/types.html#polyvartype-fn

Implicit parameters are not dynamically scoped

Implicit parameters of Haskell are often regarded as similar to dynamically scoped variables, such as 'special' variables of Common Lisp, current-input-port of Scheme, |stdin|, and exception handlers in many languages. We demonstrate on a simple example that implicit parameters lack truly dynamic scope.

http://okmij.org/ftp/Computation/dynamic-binding.html#implicit-parameter-neq-dynvar

Haskell implementation of true dynamically bound variables, via delimited continuation (monad): http://okmij.org/ftp/Computation/dynamic-binding.html#DDBinding

7.5.2 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. The question is how do these things 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.

One of the applications to the analysis of ecological transects (see below) has now been published.

A student project by James 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 multi-variate 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.

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, 31(2), pages 193–202, March–April 2007.
  http://dx.doi.org/10.1016/j.actao.2006.09.002
- 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/

7.5.3 Bioinformatics tools

Report by:	Ketil Malde

The bioinformatics stuff is developing at erratic rates, and I’m working to collect useful functions and data structures in a separate library. Currently, there is support for Fasta and TwoBit sequence formats, BLAST output. There library also contains my experiemnts in sequence aligment algorithms, and some functionality for calculating entropy, indexing based on word hashes, etc.

The library abstracts useful functionality for a handful of applications, including an EST clustering program `xsact`, a repeat detector/masker `RBR`, a tool for calculating cluster similarity with a bunch of metrics `clusc`, and Real Soon Now^{tm} a sequencing simulator supporting very general error models.

Everything is GPL and available as darcs repos (→ 6.10), at http://www.ii.uib.no/~ketil/bioinformatics/repos.

Further reading

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

7.5.4 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.

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.cofc.edu/~mitchenerg
- http://www.crouton.org