Haskell Communities and Activities Report

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

Fifteenth Edition — November 2008

Janis Voigtländer (ed.)
Peter Achten
Alfonso Acosta
Andy Adams-Moran
Lloyd Allison
Tiago Miguel Laureano Alves
Krasimir Angelov
Apfelmus
Emil Axelsson
Arthur Baars
Sengan Baring-Gould
Justin Bailey
Alistair Bayley
Jean-Philippe Bernardy
Clifford Beshers
Gwern Branwen
Joachim Breitner
Niklas Broberg
Bjorn Buckwalter
Denis Bueno
Andrew Butterfield
Roman Cheplyaka
Olaf Chitil
Jan Christiansen
Sterling Clover
Duncan Coutts
Jácome Cunha
Nils Anders Danielsson
Atze Dijkstra
Robert Dockins
Chris Eidhof
Conal Elliott
Henrique Ferreiro García
Sebastian Fischer
Leif Frenzel
Nicolas Frisby
Richard A. Frost
Peter Gavin
Andy Gill
George Giorgidze
Dimitry Golubovsky
Daniel Gorin
Jurriaan Hage
Bastiaan Heeren
Aycan Irican
Judah Jacobson
Wolfgang Jeltsch
Kevin Hammond
Enzo Haussecker
Christopher Lane Hinson
Guillaume Hoffmann
Martin Hofmann
Liyang HU
Paul Hudak
Graham Hutton
Wolfram Kahl
Garrin Kimmell
Oleg Kiselyov
Farid Karimipour
Edward Kmett
Lennart Kolmodin
Slawomir Kolodynski
Michal Konečný
Eric Kow
Stephen Lavelle
Sean Leather
Huiqing Li
Bas Lijnse
Ben Lippmeier
Andres Löh
Rita Loogen
Ian Lynagh
John MacFarlane
Christian Maeder
José Pedro Magalhães
Ketil Malde
Blažević Mario
Simon Marlow
Michael Marte
Bart Massey
Simon Michael
Arie Middelkoop
Ivan Lazar Miljenovic
Neil Mitchell
Maarten de Mol
Dino Morelli
Matthew Naylor
Jürgen Nicklisch-Franken
Rishiyur Nikhil
Thomas van Noort
Jeremy O'Donoghue
Bryan O'Sullivan
Patrick O. Perry
Jens Petersen
Simon Peyton Jones
Dan Popa
Fabian Reck
Claus Reinke
Alexey Rodriguez
Alberto Ruiz
David Sabel
Matthew Sackman
Uwe Schmidt
Tom Schrijvers
Paulo Silva
Ben Sinclair
Ganesh Sittampalam
Jim Snow
Dominic Steinitz
Don Stewart
Jon Strait
Martin Sulzmann
Doaitse Swierstra
Wouter Swierstra
Hans van Thiel
Henning Thielemann
Phil Trinder
Jared Updike
Marcos Viera
Miguel Vilaca
Janis Voigtländer
Edsko de Vries
David Waern
Jinjing Wang
Malcolm Wallace
Eelis van der Weegen
Ashley Yakeley
Brent Yorgey

Preface

This is the 15th edition of the Haskell Communities and Activities Report. There are interesting news on the implementation front, new analysis and transformation tools, many fresh projects, and new developments in established ones. Generic programming is one field that has seen a lot of activity, and you will no doubt identify other recent trends as you go through the report.

As usual, entries that are completely new (or have been revived after having disappeared temporarily) are formatted using a blue background. Updated entries have a header with a blue background. In most cases of entries that have not been changed for a year or longer, these have been dropped. Please do revive them if you have news on them for the next report.

The next report will be compiled in half a year. More details around May — watch the mailing lists for announcements. But now enjoy the report and see what other Haskellers have been up to lately. Any kind of feedback is of course very welcome ⟨hcar@haskell.org⟩.

Janis Voigtländer, Technische Universität Dresden, Germany

Contents

1 General

1.1 HaskellWiki and haskell.org

Report by:	Ashley Yakeley
Participants:	John Peterson, Olaf Chitil

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. It is one of the largest channels on freenode. The irc channel is home to hpaste and lambdabot (→ 6.12.2), two useful Haskell bots. Point your IRC client to irc.freenode.net and join the `#haskell` conversation!

For non-English conversations about Haskell there are 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` — Japanese 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 (→ 2.9.1)
- `#haskell-books` — Authors organizing the collaborative writing of the Haskell Wikibook (→ 1.5.3)
- `#darcs` — Darcs revision control channel (→ 6.1.1)
- `#ghc` — GHC developer discussion (→ 2.1)
- `#happs` — HAppS Haskell Application Server channel
- `#xmonad` — XMonad, a tiling window manager (→ 6.1.2)

Further reading

http://haskell.org/haskellwiki/IRC_channel

1.3 The Monad.Reader

Report by:	Wouter Swierstra

There are plenty of academic papers about Haskell and plenty of informative pages on the HaskellWiki (→ 1.1). Unfortunately, there is not much between the two extremes. That is 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 do not warrant an academic publication — but that does not mean these ideas are not worth writing about! Communicating ideas to a wide audience is much more important than concealing them in some esoteric journal. Even if it has 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 do not 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.

Since last year, I have moved a lot of old articles from the old MoinMoin wiki to the new MediaWiki wiki. Unfortunately, I do not have the time to reformat all the old articles. If you fancy a go at tidying an article or two, I would really appreciate your help!

I am always interested in new submissions, whether you are an established researcher or fledgling Haskell programmer. Check out the Monad.Reader homepage for all the information you need to start writing your article.

Further reading

http://www.haskell.org/haskellwiki/The_Monad.Reader

1.4 Haskell Weekly News

Report by:	Don Stewart

The Haskell Weekly News (HWN) is an irregular 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. The Haskell Weekly News also publishes latest releases uploaded to Hackage.

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

Further reading

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

1.5 Books and tutorials

1.5.1 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 include: freely accessible powerpoint slides for each chapter; solutions to exercises, and examination questions (with solutions) available to instructors; downloadable code that is compliant with the latest Haskell release.

Publication details:
- Published by Cambridge University Press, 2007. Paperback: ISBN 0521692695; Hardback: ISBN: 0521871727; eBook: ISBN 051129218X; Kindle: ASIN B001FSKE6Q.

In-depth review:
- Duncan Coutts, The Monad.Reader (→ 1.3), http://www.haskell.org/sitewiki/images/0/03/TMR-Issue7.pdf

Further reading

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

1.5.2 Real World Haskell

Report by:	Bryan O'Sullivan
Participants:	John Goerzen, Don Stewart
Status:	active development

The book "Real World Haskell" about the practical application of Haskell to everyday programming problems has been published in November 2008 by O'Reilly.

Our intended audience is programmers with no background in functional languages. We explore a diverse set of topics, among which are the following.

- Basics of Haskell and functional programming
- Developing software using standard tools like GHC and the Cabal packaging system
- Code coverage, quality assurance, and performance analysis
- Putting theory to work: working with and creating monoids, normal and applicative functors, monads, and monad transformers
- Applied topics: databases, filesystems, GUI programming, web and other network clients, web servers
- Concurrent, parallel, and transactional programming
- Error handling in pure and impure code
- Interfacing to C libraries
- Many case studies and runnable code examples

We are excited to be publishing the book under a Creative Commons License.

Further reading

http://book.realworldhaskell.org/

1.5.3 Haskell Wikibook

Report by:	Apfelmus
Participants:	Eric Kow, David House, Joeri van Eekelen, and other contributors
Status:	active development

The goal of the Haskell wikibook project is to build a community textbook about Haskell that is at once free (as in freedom and in beer), gentle, and comprehensive. We think that the many marvelous ideas of lazy functional programming can and thus should be accessible to everyone in a central place.

Recently, the wikibook has been advancing rather slowly. The rewrite of the Monad chapters is still in progress and material about lazy evaluation is still being written. Of course, additional authors and contributors that help writing new contents or simply spot mistakes and ask those questions we had never thought of are more than welcome!

Further reading

◦ http://en.wikibooks.org/wiki/Haskell
◦ Mailing list: ⟨wikibook@haskell.org⟩

1.5.4 Gtk2Hs tutorial

Report by:	Hans van Thiel

Most of the original GTK+2.0 tutorial by Tony Gail and Ian Main has been adapted to Gtk2Hs (→5.11.1), which is the Haskell binding to the GTK GUI library.

The Gtk2Hs tutorial also builds on "Programming with gtkmm" by Murray Cumming et al. and the Inti (Integrated Foundation Classes) tutorial by the Inti team.

The Gtk2Hs tutorial assumes intermediate level Haskell programming skills, but no prior GUI programming experience.

It has been translated into Spanish, by Laszlo Keuschnig, and both versions are available on Haskell darcs.

See: http://darcs.haskell.org/gtk2hs/docs/tutorial/Tutorial_Port/

1. Introduction
2. Getting Started
3. Packing
   3.1 Packing Widgets
   3.2 Packing Demonstration Program
   3.3 Packing Using Tables
4. Miscellaneous Widgets
   4.1 The Button Widget
   4.2 Adjustments, Scale, and Range
   4.3 Labels
   4.4 Arrows and Tooltips
   4.5 Dialogs, Stock Items, and Progress Bars
   4.6 Text Entries and Status Bars
   4.7 Spin Buttons
5. Aggregated Widgets
   5.1 Calendar
   5.2 File Selection
   5.3 Font and Colour Selection
   5.4 Notebook
6. Supporting Widgets
   6.1 Scrolled Windows
   6.2 EventBoxes and ButtonBoxes
   6.3 The Layout Container
   6.4 Paned Windows and Aspect Frames
7. Action Based Widgets
   7.1 Menus and Toolbars
   7.2 Popup Menus, Radio Actions, and Toggle Actions

Appendix: Drawing with Cairo: Getting Started

The Glade tutorial, an introduction to visual Gtk2Hs programming, has been updated to Glade 3 by Alex Tarkovsky. It is available on: http://haskell.org/gtk2hs/docs/tutorial/glade/ This tutorial has also been translated into Spanish, by Laszlo Keuschnig, but it is currently only available on: http://home.telfort.nl/sp969709/glade/es-index.html

1.5.5 Monad Tutorial

Report by:	Hans van Thiel
Status:	stable, might be expanded later

The "Greenhorn's Guide to becoming a Monad Cowboy" is yet another monad tutorial. It covers the basics and some examples, including a monad transformer, in a style which is a variation on the "for dummies" style. Estimated learning time is 1–2 days. It is available at http://www.muitovar.com/monad/moncow.xhtml

Further reading

http://www.muitovar.com/

1.5.6 Oleg's Mini tutorials and assorted small projects

Report by:	Oleg Kiselyov

The collection of various Haskell mini tutorials and assorted small projects (http://okmij.org/ftp/Haskell/) has received two additions:

Data-Generic and Data-Extensible Programming in Haskell

This web page describes the generic programming library "Smash" and a couple of its applications. Smash is a generic programming approach based on a type-level typecase, best understood as a static dual of "Scrap your boilerplate I" (SYB1). The Smash approach is powerful to express traversals where the type of the result is computed from the type of the transformer and the type/structure of the original term. An example is replacing all Floats with Doubles in an arbitrary term, e.g., made of Maybes, tuples, lists; the result type is computed and need not be specified.

One application explained on the web page is generic de-serialization: reconstructing a term from a flat list of its fields and a proto-type term specifying the desired structure. The Smash library is part of the extensive generic programming comparison benchmark by Alexey Rodriguez Yakushev, Alex Gerdes, and Johan Jeuring. The implementation of benchmark tests in Smash can be found at http://darcs.haskell.org/generics/comparison/SmashA/

The web page also describes a variation of the SYB3 type-class-based generic programming library that avoids both higher-rank types and mutually recursive instances. Because of the latter our code, unlike SYB3, works even in Hugs.

http://okmij.org/ftp/Haskell/generics.html

State Monad as a term algebra

We show the implementation of the state monad as a term algebra: a monadic action is a term built from sub-terms `Bind`, `Return`, `Get`, and `Put`. The constructors of the action are neither variants nor GADTs. The function `runst` (a method of the type class `RunState`) takes the initial state and the action data type, and interprets the action manipulating the state accordingly. The only non-trivial part is the interpretation of `Bind`, due to the polymorphism of the monadic bind operation. Although our implementation uses no GADTs, we nevertheless statically ensure that the interpretation of an action never gets stuck.

http://okmij.org/ftp/Haskell/types.html#state-algebra

1.5.7 Haskell Cheat Sheet

Report by:	Justin Bailey

I have created a "cheat sheet" for Haskell. It is a PDF that tries to summarize Haskell 98's syntax, keywords, and other language elements. It is built from a literate source file, so all the examples in the cheat sheet are executable. The cheatsheet is on Hackage. Once downloaded, unpack the archive and you will see the PDF and literate source.

I will be hosting the PDF directly, but for now I wanted a "limited" release. Posting to Hackage limits the audience somewhat. I will send an additional announcement when feedback has been incorporated and the PDF is available generally.

Further reading

http://hackage.haskell.org/cgi-bin/hackage-scripts/package/CheatSheet

2 Implementations

2.1 The Glasgow Haskell Compiler

Report by: Simon Peyton Jones
Participants: Geoff Mainland, Max Bolingbroke, Dan Licata, Manuel Chakravarty, David Waern, Simon Marlow, Thomas Schilling, Tim Chevalier, Roman Leshchinskiy, John Dias, Donnie Jones, Jost Berthold, Clemens Fruhwirth, and many others

For the last six months we have been primarily focused on the 6.10.1 release, which should be out by the time you read this. We are extremely grateful for the increasing support we get from the community in putting GHC releases together; more people than ever before are now helping maintain subcomponents, implementing features, fixing bugs, testing release candidates, and much more besides. We could not have made this release without your help!

The GHC 6.10 branch

GHC 6.10.1 is the first release in the 6.10 branch, and features many improvements over the 6.8 branch. The release notes have fully details, but the highlights are:

- Some new language features have been implemented:
  - Record syntax: wild-card patterns, punning, and field disambiguation
  - Generalized quasi-quotes (Geoff Mainland), from the paper Why it's nice to be quoted: quasi-quoting in Haskell (Haskell workshop 2007)
  - Generalized list comprehensions (Max Bolingbroke), from the paper Comprehensive comprehensions: comprehensions with "Order by" and "Group by" (Haskell workshop 2007)
  - View patterns (Dan Licata); see view patterns wiki page
- Type families have been completely re-implemented, by Manuel Chakravarty, along the lines of our ICFP 2008 paper Type checking with open type functions — only simpler. As a result, we believe that type families work reliably in GHC 6.10. There is one missing feature, however, namely the ability to have equalities in the superclass context of a class. We will add that to the HEAD in the next few months. An up-to-date wiki page tracks design issues and current status.
- GHC now comes with Haddock 2, which supports all GHC extensions, thanks to David Waern.
- Parallel garbage collection has been implemented by Simon Marlow. This speeds up even purely-sequential programs, by using the extra processors during garbage collection. Our ISMM'08 paper gives the details Parallel generational-copying garbage collection with a block-structured heap.
- The base library now provides, and uses, extensible exceptions, as described in Simon Marlow's paper An Extensible Dynamically-Typed Hierarchy of Exceptions (Haskell workshop 2006).
- Thomas Schilling has made the GHC API easier to use, by using a `Ghc` monad to carry the session state. Furthermore, the API now has Haddock documentation.
- External core (output only) now works again, thanks to Tim Chevalier.
- Data Parallel Haskell (DPH) comes as part of GHC, as a result of Roman Leshchinskiy's efforts. In 6.10, for the first time, DPH includes a full vectorizer, so the system is much more usable than before. It is still really an alpha release, though; we very much welcome friendly guinea pigs, but it is not ready for your 3 gigabyte genome search program. We have a lot of performance tuning to do. We have written a new paper Harnessing the multicores: nested data parallelism in Haskell (FSTTCS'08), which gives a tutorial overview of the system, focusing especially on vectorization.

The GHC 6.12 branch

Meanwhile, development goes on in the HEAD:

- John Dias has been working hard on rewriting GHC's backend, and his changes should be landing in the HEAD during October. You can find an overview of the new architecture on the wiki.
- Data Parallel Haskell remains under very active development.
- We hope that Max Bolingbroke's Dynamically Loaded Plugins summer of code project will be merged in time for 6.12. Part of this is a new, modular system for user-defined annotations, rather like Java or C# attributes. These attributes are persisted into interface files, can be examined and created by plugins, or by GHC API clients.
- Likewise, Donnie Jones' project for profiling parallel programs should be merged in time for 6.12.

◦ Simon Marlow is working on improving parallel performance, incorporating the work done by Jost Berthold during his internship at Microsoft in the summer of 2008. The plan is to make writing performant parallel programs less of a trial-and-error process, by whacking as many bottlenecks as we can find in the runtime system. We are already making significant improvements, and there is plenty more low-hanging fruit to pick. One large project that we hope to tackle is the issue of doing independent per-CPU garbage collection.

◦ Shared Libraries are inching ever closer to being completed. Clemens Fruhwirth has been working on polishing the support for shared libraries on Unix systems in particular, and when the remaining issues are ironed out we should be able to roll them out in a release.

◦ Finally, unicode text I/O and dynamic libraries were slated for 6.10 but were not quite ready in time, so we certainly expect those to make it for in 6.12.

From a development point of view, there are a couple of changes on the horizon:

◦ We plan to change how GHC's build system works, to decouple it from Cabal's internals. Our current plans are here.

◦ We plan to change from darcs to git for the version control system used by GHC; our plans are described here.

We plan to make the build-system changes first, and only then tackle the version control system.

Summary

Keeping GHC functioning for an increasingly-diverse user base is quite a challenge, especially as we keep changing the wheels while the bus is driving along. Please do consider joining in; there are plenty of things that need doing, and do not require intimate knowledge of the internals. We could particularly do with more help on supporting the Windows, Sparc, and BSD ports of GHC.

2.2 nhc98

Report by:	Malcolm Wallace
Status:	stable, maintained

nhc98 is a small, easy to install, compiler for Haskell'98. nhc98 is still very much alive and working, although it does not see many new features these days. We expect a new public release (1.22) soon, to coincide with the release of ghc-6.10.x, in particular to ensure that the included libraries are compatible across compilers.

Further reading

◦ http://haskell.org/nhc98
◦ `darcs get` http://darcs.haskell.org/nhc98

2.3 yhc

Report by:	Neil Mitchell
Participants:	Dimitry Golubovsky

The York Haskell Compiler (yhc) is a fork of the nhc98 compiler (→ 2.2), with goals such as increased portability, platform independent bytecode, integrated Hat (→ 4.3.6) 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.

There are a number of projects that make use of the Yhc.Core library, in particular a Javascript and Erlang backend.

Further reading

◦ Homepage: http://www.haskell.org/haskellwiki/Yhc
◦ Darcs repository: http://darcs.haskell.org/yhc
◦ Yhc Javascript Web Service http://www.haskell.org/haskellwiki/Yhc_web_service

2.4 The Helium compiler

Report by:	Jurriaan Hage
Participants:	Bastiaan Heeren, Arie Middelkoop

Helium is a compiler that supports a substantial subset of Haskell 98 (but, e.g., n+k patterns are missing). Type classes are restricted to a number of built-in type classes and all instances are derived. The advantage of Helium is that it generates novice friendly error feedback. The latest versions of the Helium compiler are available for download from the new website located at http://www.cs.uu.nl/wiki/Helium. This website also explains in detail what Helium is about, what it offers, and what we plan to do in the near and far future.

We are still working on making version 1.7 available, mainly a matter of updating the documentation and testing the system. Internally little has changed, but the interface to the system has been standardized, and the functionality of the interpreters has been improved and made consistent. We have made new options available (such as those that govern where programs are logged to). The use of Helium from the interpreters is now governed by a configuration file, which makes the use of Helium from the interpreters quite transparent for the programmer. It is also possible to use different versions of Helium side by side (motivated by the development of Neon (→ 5.3.6)).

A student is currently in the process of adding type class and instance definitions to the language. The work on the documentation has progressed quite a bit, but there has been little testing thus far, especially on a platform such as Windows.

2.5 EHC, “Essential Haskell” Compiler

Report by:	Atze Dijkstra
Participants:	Jeroen Fokker, Doaitse S. Swierstra, Arie Middelkoop, Lucília Camarão de Figueiredo, Carlos Camarão de Figueiredo
Status:	active development

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

For its description an Attribute Grammar system (AG) is used as well as other formalisms allowing compact notation like parser combinators. For the description of type rules, and the generation of an AG implementation for those type rules, we use the Ruler system. For source code management we use Shuffle, which allows partitioning the system into a sequence of steps and aspects. (Both Ruler and Shuffle are 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 organized 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; the last step corresponds to full Haskell.
- ◦ Independent of each step the implementation is organized into a set of aspects. Currently the type system and code generation are defined as aspects, which can then be left out so the remaining part can be used as a barebones starting point.
- ◦ Each combination of step + aspects corresponds to an actual, that is, an executable compiler. Each of these compilers is a compiler in its own right.
- ◦ 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 offers experimental implementation of more advanced features like higher-ranked polymorphism, partial type signatures, and kind polymorphism. Part of the description of the series of EH compilers is available as a PhD thesis.

What is EHC's status, what is new?

- ◦ A Haskell frontend plus Prelude has been made, compiled code runs with an interpreter. The compiler has an acceptable memory + resource footprint (done by Atze Dijkstra).
- ◦ A GRIN (Graph Reduction Intermediate Notation) based backend is available, offering global program optimization and code generation to C (done by Jeroen Fokker) as well as LLVM (done by John van Schie).
- ◦ Work has started on formalizing EHC's type system; extending our Ruler system will be part of this effort (by Lucília Camarão de Figueiredo, Carlos Camarão de Figueiredo, Arie Middelkoop, Atze Dijkstra).
- ◦ The organization of EHC into aspects, allowing better partial reuse of EHC.
- ◦ Though not a direct part of EHC, its supporting tools (AG, Shuffle) are regularly adapted to allow a cleaner EHC code base.

Is EHC used, can I use EHC? Yes, but the answer also depends for what purpose. Although it compiles a Prelude, we have yet to prepare a release of EHC as a Haskell compiler. Also, the first release will definitively be a alpha release, meant for play and experimentation, not for compiling real world programs.

EHC is used as a platform for experimentation, see EHC's webpage for various projects related to EHC. EHC can be downloaded from our svn repository.

What will happen with EHC in the near future? We plan to do the following:

- ◦ Make the variant for full Haskell available as a Haskell compiler. For this we will stabilize the implementation and add proper documentation.
- ◦ Rework the type system to have a more formal underpinning. Our intent is to use and extend our Ruler system for this.

Further reading

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

2.6 Hugs as Yhc Core Producer

Report by:	Dimitry Golubovsky
Status:	Experimental

Background

Hugs is one of the oldest implementations of Haskell known, an interactive compiler and bytecode interpreter. Yhc (→ 2.3) is a fork of nhc98 (→ 2.2). Yhc Core is an intermediate form Yhc uses to represent a compiled Haskell program.

Yhc converts each Haskell module to a binary Yhc Core file. Core modules are linked together, and all redundant (unreachable) code is removed. The Linked Core is ready for further conversions by backends.

Hugs loads Haskell modules into memory and stores them in a way to some degree similar to Yhc Core. Hugs is capable to dump its internal storage structure in textual form (let us call it Hugs Core). The output looks similar to Yhc Core, pretty-printed. This was initially intended for debugging purposes, however several Hugs CVS (now darcs) log records related such output to some "Snowball Haskell compiler" ca. 2001.

The experiment

The goal of the experiment described here was to convert Hugs Core into Yhc Core, so Hugs might become a frontend for existing and future Yhc Core optimizers and backends. At least one benefit is clear: Hugs is well maintained to be compatible with recent versions of Haskell libraries and supports many of Haskell language extensions that Yhc does not yet support.

The necessary patches were pushed to the main Hugs repository in June 2008, thanks to Ross Paterson for reviewing them. The following changes were made:

1. A configuration option was added to enable the generation of Hugs Core.

2. The toplevel Makefile was modified to build an additional executable, `corehugs`.

3. Consistency of Hugs Core output in terms of naming of modules and functions was improved.

The `corehugs` program converts Haskell source files into Hugs Core files, one for one. All functions and data constructors are preserved in the output, whether reachable or not. Unreachable items will be removed later using Yhc Core tools.

The conversion of Hugs Core to Yhc Core is performed outside of Hugs using the `hugs2yc` package. The package provides a parser for the syntax of Hugs Core and an interface to the Yhc Core Linker. All Hugs Core files written by `corehugs` are read in and parsed, resulting in the set of Yhc Core modules in memory. The modules are linked together using the Yhc Core Linker, and all unreachable items are removed at this point. A "driver" program that uses the package may save the linked Yhc Core in a file, or pass it on to a backend. The code of the `hugs2yc` package is compatible to both Hugs and GHC.

Availability

In order to use the new Hugs functionality, obtain Hugs from the "HEAD" darcs repo, see http://hackage.haskell.org/trac/hugs/wiki/GettingTheSource. However, Hugs obtained in such a way may not always compile. This Google Code project: http://code.google.com/p/corehugs/ hosts specialized snapshots of Hugs that are more likely to build on a random computer and also include additional packages necessary to work with Yhc Core.

Future plans

Further effort will be taken to standardize various aspects of Yhc Core, especially the specification of primitives, because all backends must implement them uniformly. This Google spreadsheet: http://tinyurl.com/prim-normal-set contains the proposal for an unified set of Yhc Core primitives.

Work is in progress on various backends for Yhc Core, including Javascript, Erlang, Python, JVM, .NET, and others. This Wiki page: http://tinyurl.com/ycore-conv-infra summarizes their development status.

Further reading

- Yhc Core conversion infrastructure
  http://tinyurl.com/ycore-conv-infra
- Download Hugs specialized snapshots
  http://code.google.com/p/corehugs/
- Proposed specification of the Normal Set of primitives
  http://tinyurl.com/prim-normal-set
- A brief example of using `corehugs`
  http://code.google.com/p/corehugs/wiki/Demonstration

2.7 Haskell frontend for the Clean compiler

Report by:	Thomas van Noort
Participants:	John van Groningen, Rinus Plasmeijer
Status:	active development

We are currently working on a frontend for the Clean compiler (→ 3.2.3) that supports a subset of Haskell 98. This will allow Clean modules to import Haskell modules, and vice versa. Furthermore, we will be able to use some of Clean's features in Haskell code, and vice versa. For example, we could define a Haskell module which uses Clean's uniqueness typing, or a Clean module which uses Haskell's newtypes. The possibilities are endless!

Future plans

We hope to release a beta version of the new Clean compiler, solely to the institution in Nijmegen, by the end of this year. But there is still a lot of work to do before we are able to release it to the outside world, so we cannot make any promises regarding the release date. Keep an eye on the Clean mailing lists for any important announcements!

Further reading

http://wiki.clean.cs.ru.nl/Mailing_lists

2.8 The Reduceron

Report by:	Matthew Naylor
Participants:	Colin Runciman
Status:	Experimental

The Reduceron is a prototype of a special-purpose graph reduction machine, built using an FPGA. It can access up to eight graph nodes in parallel on each of its stack, heap, and combinator memories. The goal so far has been to optimize function application. Eight combinator nodes can be instantiated with eight stack elements and placed on the heap, all in a single cycle.

The Reduceron is a simple machine, containing just four instructions and a garbage collector, and executes core Haskell almost directly. The translator to bytecode and the FPGA machine are both implemented in Haskell, the latter using Lava (→ 6.9.2). See the URL below for details and results.

Since the last HCAR, I have written a thesis in which chapter 2 is dedicated to the Reduceron. I am now working on a new Reduceron which I hope will exploit wide, parallel memories further. I am also working on a new variant of Lava, to support the demands of the Reduceron.

Further reading

http://www.cs.york.ac.uk/~mfn/reduceron2/

2.9 Platforms

2.9.1 Haskell in Gentoo Linux

Report by:	Lennart Kolmodin

GHC version 6.8.2 has been in Gentoo since late last year, and is about to go stable. All of the 60+ Haskell libraries and tools work with it, too. There are also GHC binaries available for alpha, amd64, hppa, ia64, sparc, and x86.

Browse the packages in portage at http://packages.gentoo.org/category/dev-haskell?full_cat.

The GHC architecture/version matrix is available at http://packages.gentoo.org/package/dev-lang/ghc.

Please report problems in the normal Gentoo bug tracker at bugs.gentoo.org.

There is also a Haskell overlay providing another 200 packages. Thanks to the recent progress of Cabal and Hackage (→ 5.1), we have written a tool called "hackport" (initiated by Henning Günther) to generate Gentoo packages that rarely need much tweaking.

The overlay is available at http://haskell.org/haskellwiki/Gentoo. Using Darcs (→ 6.1.1), it is easy to keep updated and send patches. It is also available via the Gentoo overlay manager "layman". If you choose to use the overlay, then problems should be reported on IRC (`#gentoo-haskell` on freenode), where we coordinate development, or via email ⟨haskell@gentoo.org⟩.

Lately a few of our developers have shifted focus, and only a few developers remain. If you would like to help, which would include working on the Gentoo Haskell framework, hacking on hackport, writing ebuilds, and supporting users, please contact us on IRC or email as noted above.

2.9.2 Fedora Haskell SIG

Report by:	Jens Petersen
Participants:	Bryan Sullivan, Yaakov Nemoy, Fedora Haskell SIG
Status:	on-going

The Fedora Haskell SIG is an effort to provide good support for Haskell in Fedora.

We now have a set of rpm macros and Packaging Guidelines for packaging Cabal-based packages in Fedora: so it is now fairly easy to get Haskell packages reviewed and approved by package reviewers in Fedora.

Fedora 10 will ship with ghc-6.8.3 and the new rpm macros at the end of this month.

For Fedora 11 we are planning to move to ghc-6.10 and add plenty of Haskell libraries using the new Fedora Haskell Packaging Guidelines, and hopefully also experiment with shared libraries and cabal-install.

Further reading

http://fedoraproject.org/wiki/SIGs/Haskell

3 Language

3.1 Extensions of Haskell

3.1.1 Haskell Server Pages (HSP)

Report by:	Niklas Broberg
Status:	active development

Haskell Server Pages (HSP) is an extension of Haskell targeted at writing dynamic web pages. Key features and selling points include:

- Use literal XML syntax in your Haskell code for creating values of appropriate datatypes. (Note though that writing literal XML is quite optional, if you, like me, do not really enjoy that language.)
- Guarantees that XML output is well-formed (and an HTML output mode if that is what you need).
- A model that gives easy access to necessary environment variables.
- Simple programming model that is easy to use even for non-experienced Haskell programmers, in particular with a very simple transition from static XML pages to dynamic HSP pages.
- Easy integration with a DSL called HJScript that makes it easy to write client-side (JavaScript) scripts.
- An extension of HAppS that can serve HSP pages on the fly, making deployment of pages really simple.

HSP is continuously released onto Hackage. It consists of a series of interdependent packages with package hsp as the main top-level starting point, and package happs-hsp for integration with HAppS. The best way to keep up with development is to grab the darcs repositories, all located under http://code.haskell.org/HSP.

Further reading

http://haskell.org/haskellwiki/HSP

3.1.2 GpH — Glasgow Parallel Haskell

Report by:	Phil Trinder
Participants:	Abyd Al Zain, Mustafa Aswad, Jost Berthold, Jamie Gabbay, Murray Gross, Hossein Haeri, Kevin Hammond, Vladimir Janjic, Hans-Wolfgang Loidl

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

- Both GpH and Eden parallel Haskells are being used for parallel language research and in the SCIEnce project (see below).
- We are making comparative evaluations of a range of GpH implementations and other parallel functional languages (Eden and Feedback Directed Implicit Parallelism (FDIP)) on multicore architectures.
- We are teaching parallelism to undergraduates using GpH at Heriot-Watt and Phillips Universität Marburg.
- We are developing a big step operational semantics for `seq` and using it to prove identities.

GpH Applications

As part of the SCIEnce EU FP6 I3 project (026133) (→ 8.6) (April 2006 – April 2011) we use GpH and Eden as middleware to provide access to computational grids from Computer Algebra(CA) systems, including GAP, Maple MuPad and KANT. We have designed, implemented and are evaluating the SymGrid-Par interface that facilitates the orchestration of computational algebra components into high-performance parallel applications.

In recent work we have demonstrated that SymGrid-Par is capable of exploiting a variety of modern parallel/multicore architectures without any change to the underlying CA components; and that SymGrid-Par is capable of orchestrating heterogeneous computations across a high-performance computational Grid.

Implementations

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

- The focus of the development has switched to versions tracking GHC releases, currently GHC 6.8, and the development version is available upon request to the GpH mailing list (see the GpH web site).
- 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.

We are exploring new, *prescient* scheduling mechanisms for GpH.

Our main hardware platforms are Intel-based Beowulf clusters and multicores. 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⟩

3.1.3 Eden

Report by:	Rita Loogen
Participants:	**in Madrid:** Ricardo Peña, Yolanda Ortega-Mallén, Mercedes Hidalgo, Fernando Rubio, Alberto de la Encina, Lidia Sánchez-Gil **in Marburg:** Jost Berthold, Mischa Dieterle, Oleg Lobachev, Thomas Horstmeyer, Johannes May
Status:	Ongoing

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), synchronization, 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 parallelize 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 6.8.1 is available from the Marburg group on request. Support for Glasgow parallel Haskell (→ 3.1.2) 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. Program executions can be visualized using the Eden trace viewer tool EdenTV. Recent results show that the system behaves equally well on workstation clusters and on multi-core machines.

Recent Theses

- Jost Berthold: *Implicit and Explicit Parallel Functional Programming: Concepts and Implementation*, Dissertation (PhD thesis), Fachbereich Mathematik und Informatik, Philipps-Universität Marburg, June 2008.
- Alberto de la Encina: *Formalizando el proceso de depuración en programación funcional paralela y perezosa*, Tesis Doctoral (PhD thesis), Facultad de Ciencias Matemáticas, Universidad Complutense de Madrid, June 2008, in Spanish.
- Lidia Lidia Sánchez-Gil: *Sobre la equivalencia entre semánticas operacionales y denotacionales para lenguajes funcionales paralelos*, Master Thesis, Universidad Complutense de Madrid, September 2008 (in Spanish).

Recent and Forthcoming Publications

- Jost Berthold, Simon Marlow, Kevin Hammond, and Abyd Al Zain: *Comparing and Optimising Parallel Haskell Implementations on Multicore*, Draft Proceedings of the 20th International Symposium on the Implementation and Application of Functional Languages (IFL), September 2008
- Mischa Dieterle, Jost Berthold, and Rita Loogen: *Functional Skeleton Implementations for Parallel Map-and-Reduce*, Draft Proceedings of the 20th International Symposium on the Implementation and Application of Functional Languages (IFL), September 2008.
- Jost Berthold, Mischa Dieterle, and Rita Loogen: *A Distributed Work Pool Skeleton in Eden*, submitted.
- Oleg Lobachev, Jost Berthold, Mischa Dieterle, and Rita Loogen: *Parallel FFT With Eden Skeletons*, in preparation.
- Oleg Lobachev and Rita Loogen: *Towards an Implementation of a Computer Algebra System in a Functional Language*, 9th International Conference on Artificial Intelligence and Symbolic Computa-

tion (AISC), Birmingham, July 2008, Springer LNAI 5144, 141–154.

◦ Alberto de la Encina, Ismael Rodríguez, and Fernando Rubio: *A Debugger for Parallel Haskell Dialects*, LNCS 5022, Springer 2008, 282-293.

Further reading

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

3.1.4 XHaskell project

Report by:	Martin Sulzmann
Participants:	Kenny Zhuo Ming Lu

XHaskell is an extension of Haskell which combines parametric polymorphism, algebraic data types, and type classes with XDuce style regular expression types, subtyping, and regular expression pattern matching. The latest version can be downloaded via http://code.google.com/p/xhaskell/

Latest developments

Kenny's thesis will be available by the end of the year, describing in detail the formal underpinnings behind XHaskell.

One of the things we will be looking into in the future is to turn XHaskell into a library (rather than stand-alone compiler).

3.1.5 HaskellActor (previously: HaskellActorJoin)

Report by:	Martin Sulzmann

The focus of the HaskellActor project is on Erlang-style concurrency abstractions. See for details: http://sulzmann.blogspot.com/2008/10/actors-with-multi-headed-receive.html

Novel features of HaskellActor include

◦ Multi-headed receive clauses, with support for

◦ guards, and

◦ propagation

Latest developments

The HaskellActor implementation (as a library extension to Haskell) is available via http://hackage.haskell.org/cgi-bin/hackage-scripts/package/actor

3.2 Related Languages

3.2.1 Curry

Report by:	Jan Christiansen
Participants:	Bernd Braßel, Michael Hanus, Wolfgang Lux, Sebastian Fischer, and others
Status:	active development

Curry is a functional logic programming language with Haskell syntax. In addition to the standard features of functional programming like higher-order functions and lazy evaluation, Curry supports features known from logic programming. This includes programming with non-determinism, free variables, constraints, declarative concurrency, and the search for solutions. Although Haskell and Curry share the same syntax, there is one main difference with respect to how function declarations are interpreted. In Haskell the order in which different rules are given in the source program has an effect on their meaning. In Curry, in contrast, the rules are interpreted as *equations*, and overlapping rules induce a non-deterministic choice and a search over the resulting alternatives. Furthermore, Curry allows to call functions with free variables as arguments so that they are bound to those values that are demanded for evaluation, thus providing for function inversion.

There are three major implementations of Curry. While the original implementation PAKCS (Portland Aachen Kiel Curry System) compiles to Prolog, MCC (Münster Curry Compiler) generates native code via a standard C compiler. The Kiel Curry System (KiCS) compiles Curry to Haskell aiming to provide nearly as good performance for the purely functional part as modern compilers for Haskell do. From these implementations only MCC will provide type classes in the near future. Type classes are not part of the current definition of Curry, though there is no conceptual conflict with the logic extensions.

Recent research aims at simplifying the compilation scheme of KiCS which allows for using optimizations when compiling the generated Haskell code. First tests show that this significantly improves the performance of Curry programs.

There have been research activities in the area of functional logic programming languages for more than a decade. Nevertheless, there are still a lot of interesting research topics regarding more efficient compilation techniques and even semantic questions in the area of language extensions like encapsulation and function patterns. Besides activities regarding the language itself, there is also an active development of tools concerning Curry (e.g., the documentation tool CurryDoc, the analysis environment CurryBrowser, the observation debuggers COOSy and iCODE, the debugger B.I.O. (http://www-ps.informatik.uni-kiel.de/currywiki/tools/oracle_debugger), EasyCheck (→ 4.3.2), and CyCoTest (→ 4.3.4)). Because Curry has a functional subset, these tools can canonically be transferred to the functional world.

Further reading

- http://www.curry-language.org/
- http://wiki.curry-language.org/

3.2.2 Agda

Report by:	Nils Anders Danielsson
Participants:	Ulf Norell and many others
Status:	Actively developed

Do you crave for highly expressive types, but do not want to resort to type-class hackery? Then Agda might provide a view of what the future has in store for you.

Agda is a dependently typed functional programming language (developed using Haskell). The language has inductive families, i.e. GADTs which can be indexed by *values* and not just types. Other goodies include parameterized modules, mixfix operators, and an *interactive* Emacs interface (the type checker can assist you in the development of your code).

A lot of work remains in order for Agda to become a full-fledged programming language (good libraries, mature compilers, documentation, etc.), but already in its current state it can provide lots of fun as a platform for experiments in dependently typed programming.

New since last time:

- Coinductive types (types with possibly infinite values).
- Case-split: The user interface can replace a pattern variable with the corresponding constructor patterns. You get one new left-hand side for every possible constructor.
- The foreign function interface now ensures that the foreign (Haskell) code has types matching the Agda code.
- Sized types, which can make it easier to explain why your code is terminating, are currently being implemented by Ulf Norell and Andreas Abel.
- Agda packages for Debian/Ubuntu have been prepared by Liyang HU, and Kuragaki-san has constructed a new Agda installer for Windows.
- A new Emacs input method, which contains bindings for many Unicode symbols, has been implemented by Nils Anders Danielsson.

Further reading

The Agda Wiki: http://www.cs.chalmers.se/~ulfn/Agda/

3.2.3 Clean

Participants:	Software Technology Research Group at Radboud University Nijmegen

Clean is a general purpose, state-of-the-art, pure and lazy functional programming language designed for making real-world applications. Clean is the only functional language in the world which offers *uniqueness typing*. This type system makes it possible in a pure functional language to incorporate destructive updates of arbitrary data structures (including arrays) and to make direct interfaces to the outside imperative world.

Here is a short list with notable features:

- Clean is a lazy, pure, higher-order functional programming language with explicit graph rewriting semantics.
- Although Clean is by default a lazy language one can smoothly turn it into a strict language to obtain optimal time/space behavior: functions can be defined lazy as well as (partially) strict in their arguments; any (recursive) data structure can be defined lazy as well as (partially) strict in any of its arguments.
- Clean is a strongly typed language based on an extension of the well-known Milner/Hindley/Mycroft type inferencing/checking scheme including the common higher-order types, polymorphic types, abstract types, algebraic types, synonym types, and existentially quantified types.
- Type classes and type constructor classes are provided to make overloaded use of functions and operators possible.
- Clean offers records and (destructively updateable) arrays and files.
- Clean has pattern matching, guards, list comprehensions and array comprehensions and a lay-out sensitive mode.
- The uniqueness type system makes it possible to develop efficient applications. In particular, it allows a refined control over the single threaded use of objects. Thus can influence the time and space behavior of programs. The uniqueness type system can be also used to incorporate destructive updates of objects within a pure functional framework. It allows destructive transformation of state information and enables efficient interfacing to the non-functional world (to C but also to I/O systems like X-Windows) offering direct access to file systems and operating systems.
- Clean offers a sophisticated I/O library with which window based interactive applications (and the handling of menus, dialogs, windows, mouse, keyboard, timers, and events raised by sub-applications) can

be specified compactly and elegantly on a very high level of abstraction.

- GUI-based programs written in Clean using the 0.8 I/O library can be ported without modification of source code to any one of the many platforms we support.
- There are many libraries available offering additional functionality.

Further reading

http://clean.cs.ru.nl/

3.3 Type System / Program Analysis

3.3.1 Uniqueness Typing

Report by:	Edsko de Vries
Participants:	Rinus Plasmeijer, David M Abrahamson
Status:	Completed (thesis submitted)

An important feature of pure functional programming languages is definiteness: if the same expression is used in multiple places, it must have the same value every time. A consequence of definiteness (sometimes also referred to as referential transparency) is that functions must not 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* (→3.2.3) but with various improvements: no subtyping is required, the type language does not include inequality constraints (types in Clean often involve implications between uniqueness attributes), and types and uniqueness attributes are both considered types (albeit of different kinds). This makes the type system sufficiently similar to standard Hindley/Milner type systems that (1) standard inference algorithms can be applied, and (2) modern extensions such as arbitrary rank types and generalized algebraic data types (GADTs) can easily be incorporated.

Although our type system is inspired by Clean, it is also relevant to Haskell, because the core uniqueness type system we propose is very similar to Haskell's core type system.

Further reading

- Edsko de Vries, "Making Uniqueness Typing Less Unique", PhD thesis, *forthcoming.*
- Edsko de Vries, Rinus Plasmeijer, and David Abrahamson, "Uniqueness Typing Simplified", in *Olaf Chitil, Zoltán Horváth and Viktória Zsók (Eds.): IFL 2007, LNCS 5083.*
- Edsko de Vries, Rinus Plasmeijer, and David Abrahamson, "Uniqueness Typing Redefined", in *Z. Horváth, V. Zsók, and Andrew Butterfield (Eds.): IFL 2006, LNCS 4449.*

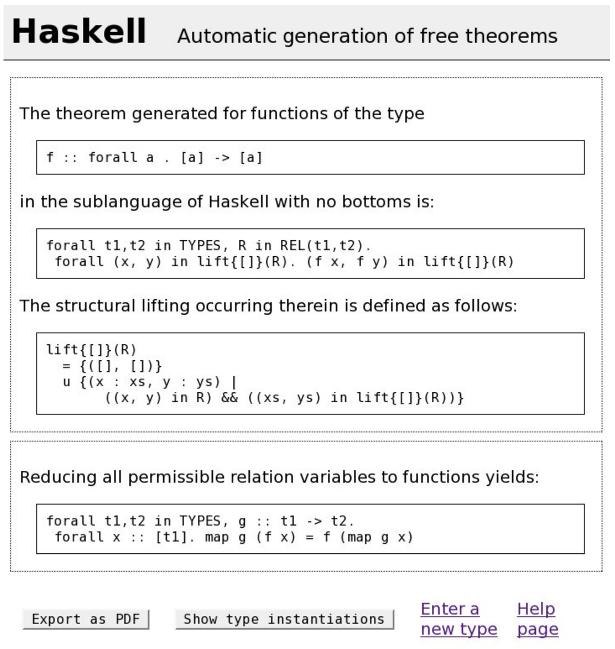
3.3.2 Free Theorems for Haskell

Report by:	Janis Voigtländer
Participants:	Florian Stenger, Daniel Seidel, Joachim Breitner

Free theorems are statements about program behavior derived from (polymorphic) types. Their origin is the polymorphic lambda-calculus, but they have also been applied to programs in more realistic languages like Haskell. Since there is a semantic gap between the original calculus and modern functional languages, the underlying theory (of relational parametricity) needs to be refined and extended. We aim to provide such new theoretical foundations, as well as to apply the theoretical results to practical problems. A recent application paper is "Bidirectionalization for Free!" (POPL'09).

Also on the practical side, we maintain a library and tools for generating free theorems from Haskell types, originally implemented by Sascha Böhme. Both the library and a shell-based tool are available from Hackage (as free-theorems and ftshell, respectively). There is also a web-based tool at http://linux.tcs.inf.tu-dresden.de/~voigt/ft. General features include:

- three different language subsets to choose from
- equational as well as inequational free theorems
- relational free theorems as well as specializations down to function level
- support for algebraic data types, type synonyms and renamings, type classes



The theorem generated for functions of the type

```
f :: forall a . [a] -> [a]
```

in the sublanguage of Haskell with no bottoms is:

```
forall t1,t2 in TYPES, R in REL(t1,t2).
 forall (x, y) in lift{[]}(R). (f x, f y) in lift{[]}(R)
```

The structural lifting occurring therein is defined as follows:

```
lift{[]}(R)
  = {([], [])}
  u {(x : xs, y : ys) |
         ((x, y) in R) && ((xs, ys) in lift{[]}(R))}
```

Reducing all permissible relation variables to functions yields:

```
forall t1,t2 in TYPES, g :: t1 -> t2.
 forall x :: [t1]. map g (f x) = f (map g x)
```

While the web-based tool is restricted to algebraic data types, type synonyms, and type classes from Haskell standard libraries, the shell-based tool also enables the user to declare their own algebraic data types and so on, and then to derive free theorems from types involving those. A distinctive feature of the web-based tool is to export the generated theorems in PDF format.

Joachim Breitner visited us in Dresden for two very productive weeks. Among other things, he implemented new post-simplifications for the free theorems generator, hopefully to be included in the web-based tool in the near future. He also wrote a web-based interface to the library from the POPL'09 paper, accessible at http://linux.tcs.inf.tu-dresden.de/~bff/cgi-bin/bff.cgi.

Further reading

http://wwwtcs.inf.tu-dresden.de/~voigt/project/

3.3.3 The Disciplined Disciple Compiler (DDC)

Report by:	Ben Lippmeier
Status:	alpha, active

Disciple is an explicitly lazy dialect of Haskell which is being developed as part of my PhD project into effect typing, optimization, and methods for combining strict and lazy evaluation in the same language.

Effect typing is offered as a practical alternative to state monads, and we suggest that state and destructive update are useful enough to deserve direct attention by the language and type system.

Disciple's type system is similar to that used in Haskell 98, with the addition of region, effect and closure information which is used to model the aliasing, side effect and data sharing properties of functions. This extra information is present in the source types, but can be fully reconstructed and does not usually place a burden on the programmer. The information is also present in DDC's core language, and is used to guide code transformation style optimizations in the presence of side effects. When the type system proves that a particular expression is visibly pure, the full gamut of optimizations can be applied.

The system also supports region, effect and closure class constraints which are modeled after the (value) type constraints of Haskell. A function's type signature can use these constraints to require certain objects to be mutable, or certain function arguments to be pure. Disciple also supports type directed field projections (i.e., record syntax), and lazy code can be seamlessly integrated with strict code without changing the shape of types, or requiring explicit forcing by the programmer.

DDC is in alpha release and comes with some cute example programs including a graphical n-body simulation, a collision detection demo, a ray-tracer, and some animated fractals. As I am currently writing up my PhD thesis, due end of December 2008, work on DDC has stalled for now. Development is likely to resume in 2nd quarter 2009. Although DDC is a full working system, it has been primarily a research vehicle so far and contains lots of cosmetic bugs. It is not yet "industrial strength".

DDC is open source and available from http://www.haskell.org/haskellwiki/DDC. If you would like to help out, then a detailed bug list is at http://code.google.com/p/disciple. There are many interesting lines of research in effect typing, the language is default strict, and if you squint it looks just like Haskell code.

4 Tools

4.1 Scanning, Parsing, Transformations

4.1.1 Alex version 2

Report by:	Simon Marlow
Status:	stable, maintained

Alex is a lexical analyzer 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 analyzer generator is often used in conjunction with a parser generator, such as Happy (→4.1.2), to build a complete parser.

The latest release is version 2.3, released October 2008. Alex is in maintenance mode, we do not anticipate any major changes in the near future.

Changes in version 2.3 vs. 2.2:

- Works with GHC 6.10.1 and Cabal 1.6.
- Support for efficient lexing of strict bytestrings, by Don Stewart.
- The `monadUserState` wrapper type was added by Alain Cremieux.

Further reading

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

4.1.2 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 1.18.2, released 5 November 2008.

Changes in version 1.18.2 vs. 1.17:

- Macro-like parameterized rules were added by Iavor Diatchki.
- Works with GHC 6.10.1 and Cabal 1.6.
- A couple of minor bugfixes: Happy does not get confused by Template Haskell quoted names in code, and a multi-word token type is allowed.

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/.

4.1.3 UUAG

Report by:	Arie Middelkoop
Participants:	ST Group of Utrecht University
Status:	stable, maintained

UUAG is the *Utrecht University Attribute Grammar* system. It is a preprocessor for Haskell which makes it easy to write *catamorphisms* (that is, functions that do to any datatype what *foldr* does to lists). You can define tree walks using the intuitive concepts of *inherited* and *synthesized attributes*, while keeping the full expressive power of Haskell. The generated tree walks are *efficient* in both space and time.

New features are support for polymorphic abstract syntax and higher-order attributes. With polymorphic abstract syntax, the type of certain terminals can be parameterized. Higher-order attributes are useful to incorporate computed values as subtrees in the AST.

The system is in use by a variety of large and small projects, such as the Haskell compiler EHC, the editor Proxima for structured documents, the Helium compiler (→2.4), the Generic Haskell compiler, and UUAG itself. The current version is 0.9.6 (April 2008), is extensively tested, and is available on Hackage.

We are currently improving the documentation, and plan to introduce an alternative syntax that is closer to the Haskell syntax.

Further reading

- http://www.cs.uu.nl/wiki/bin/view/HUT/AttributeGrammarSystem
- http://hackage.haskell.org/cgi-bin/hackage-scripts/package/uuagc-0.9.6

4.2 Documentation

4.2.1 Haddock

Report by:	David Waern
Status:	experimental, 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 (→ 5.1), 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 version 2.2.2, released August 5 2008.

Recent changes:

- Support for GHC 6.8.3
- The Hoogle backend is back, thanks to Neil Mitchell.
- Show associated types in the documentation for class declarations
- Show associated types in the documentation for class declarations
- Show type family declarations
- Show type equality predicates
- Major bug fixes (#1 and #44)
- It is no longer required to specify the path to GHC's lib dir
- Remove unnecessary parenthesis in type signatures

Future plans

Currently, Haddock ignores comments on some language constructs like GADTs and Associated Type synonyms. Of course, the plan is to support comments for these constructs in the future. Haddock is also slightly more picky on where to put comments compared to the 0.x series. We want to fix this as well. Both of these plans require changes to the GHC parser. We want to investigate to what degree it is possible to decouple comment parsing from GHC and move it into Haddock, to not be bound by GHC releases.

Other things we plan to add in future releases:

- Support for GHC 6.10.1
- HTML frames (á la Javadoc)
- Support for documenting re-exports from other packages

Further reading

- Haddock's homepage: http://www.haskell.org/haddock/
- Haddock's developer WiKi and Trac: http://trac.haskell.org/haddock

4.2.2 lhs2TEX

Report by:	Andres Löh
Status:	stable, maintained

This tool by Ralf Hinze and Andres Löh is a preprocessor that transforms literate Haskell code into LaTeX documents. The output is highly customizable by means of formatting directives that are interpreted by lhs2TEX. Other directives allow the selective inclusion of program fragments, so that multiple versions of a program and/or document can be produced from a common source. The input is parsed using a liberal parser that can interpret many languages with a Haskell-like syntax, and does not restrict the user to Haskell 98.

The program is stable and can take on large documents.

Since the last report, version 1.14 has been released. This version is compatible with (and requires) Cabal 1.6. Apart from minor bugfixes, experimental support for typesetting Agda (→ 3.2.2) programs has been added.

Further reading

http://www.cs.uu.nl/~andres/lhs2tex

4.3 Testing, Debugging, and Analysis

4.3.1 SmallCheck and Lazy SmallCheck

Report by:	Matthew Naylor
Participants:	Fredrik Lindblad, Colin Runciman
Status:	active development

SmallCheck is a one-module lightweight testing library. It adapts QuickCheck's ideas of type-based generators for test data and a class of testable properties. But instead of testing a sample of randomly generated values, it tests properties for all the finitely many values up to some depth, progressively increasing the depth used. Among other advantages, existential quantification is supported, and generators for user-defined types can follow a simple pattern and are automatically derivable.

Lazy SmallCheck is like SmallCheck, but generates partially-defined inputs that are progressively refined as demanded by the property under test. The key observation is that if a property evaluates to True or False for a partially-defined input then it would also do so for all refinements of that input. By not generating such refinements, Lazy SmallCheck may test the same input-space as SmallCheck using significantly fewer tests. Lazy SmallCheck's interface is a subset of SmallCheck's, often allowing the two to be used interchangeably.

Since the last HCAR, we have written a paper about SmallCheck and Lazy SmallCheck and we have released versions 0.4 and 0.3 respectively on Hackage. We have also squashed a bug in the Hugs implementation of exception-handling which made Lazy SmallCheck sometimes report "Control stack overflow" (Hugs bug #84). And we have found a simple way to do demand-driven generation of (first-order) functions in Lazy SmallCheck, re-using the existing machinery for demand-driven generation of data. The next release of Lazy SmallCheck will incorporate this idea, and will hopefully support existential quantification too. We are still interested in improving and harmonizing the two libraries and welcome comments and suggestions from users.

Further reading

http://www.cs.york.ac.uk/fp/smallcheck/

4.3.2 EasyCheck

Report by:	Jan Christiansen
Participants:	Sebastian Fischer
Status:	experimental

EasyCheck is an automatic test tool like QuickCheck or SmallCheck (→ 4.3.1). It is implemented in the functional logic programming language Curry (→ 3.2.1). Although simple test cases can be generated from nothing but type information in all mentioned test tools, users have the possibility to define custom test-case generators — and make frequent use of this possibility. Nondeterminism — the main extension of functional-logic programming over Haskell — is an elegant concept to describe such generators. Therefore it is easier to define custom test-case generators in EasyCheck than in other test tools. If no custom generator is provided, test cases are generated by a free variable which non-deterministically yields all values of a type. Moreover, in EasyCheck, the enumeration strategy is independent of the definition of test-case generators. Unlike QuickCheck's strategy, it is complete, i.e., every specified value is eventually enumerated if enough test cases are processed, and no value is enumerated twice. SmallCheck also uses a complete strategy (breadth-first search) which EasyCheck improves w.r.t. the size of the generated test data. EasyCheck is distributed with the Kiel Curry System (KiCS).

Further reading

http://www-ps.informatik.uni-kiel.de/currywiki/tools/easycheck

4.3.3 checkers

Report by:	Conal Elliott
Status:	active development

Checkers is a library for reusable QuickCheck properties, particularly for standard type classes (class laws and class morphisms). For instance, much of Reactive (→ 6.5.2) can be specified and tested using just these properties. Checkers also lots of support for randomly generating data values.

For the past few months, this work has been graciously supported by Anygma.

Further reading

http://haskell.org/haskellwiki/checkers

4.3.4 CyCoTest

Report by:	Sebastian Fischer
Participants:	Herbert Kuchen
Status:	experimental

The Curry Coverage Tester CyCoTest (pronounced like psycho test) aims at testing declarative programs to the bone. Unlike black-box test tools like QuickCheck, it does not generate test cases from type information or additional specifications. It rather uses the demand of the program under test to *narrow* test cases lazily. Narrowing is a generalization of reduction that allows to compute with partial information. Evaluating a program with narrowing and initially uninstantiated input binds the input as much as demanded by the computation and non-deterministically computes a corresponding result for each binding. The generated pairs of in- and output form a set of test cases that reflects the demand of the tested program.

The generated set of test cases can either be checked by hand or using properties, i.e., functions with a Boolean result. Using properties is convenient, but sometimes it is hard to come up with a complete formal specification of the tested program. Hence, errors might remain undetected if an incomplete property is used to evaluate the test cases. In order to lower the burden of manual checking, we employ control- and data-flow coverage information to minimize the set of generated test cases. Test cases that do not cause new code coverage are considered redundant and need not be shown to the user. Although this bears the risk of eliminating test cases that expose a bug, experiments indicate that the employed coverage criteria suffice to expose bugs in practice.

CyCoTest is implemented in and for the functional logic programming language Curry (→ 3.2.1), which provides narrowing for free. A Haskell implementation would be possible using ideas from the Kiel Curry System (KiCS), which translates Curry programs into Haskell programs.

Further reading

http://www-ps.informatik.uni-kiel.de/currywiki/tools/cycotest

4.3.5 G∀st

Report by:	Peter Achten
Participants:	Pieter Koopman
Status:	stable, maintained

G∀st is a fully automatic test system, written in Clean (→3.2.3). Given a logical property, stated as a function, it is able to generate appropriate test values, to execute tests with these values, and to evaluate the results of these tests. In this respect G∀st is similar to Haskell's QuickCheck.

Apart from testing logical properties, G∀st is able to test state based systems. In such tests, an extended state machine (esm) is used instead of logical properties. This gives G∀st the possibility to test properties in a way that is somewhat similar to model checking and allows you to test interactive systems, such as web pages or GUI programs. In order to validate and test the quality of the specifying extended state machine, the esmViz tool simulates the state machine and tests properties of this esm on the fly.

G∀st is based on the generic programming techniques of Clean which are very similar to Generic Haskell. G∀st is distributed as a library in the standard Clean distribution. This version is somewhat older than the version described in recent papers.

Future plans

We would like to determine the quality of the tests for instance by determining the coverage of tests. As a next step we would like to use techniques from model checking to direct the testing based on esms in G∀st.

Further reading

- http://www.cs.ru.nl/~pieter/gentest/gentest.html
- Papers on G∀st: http://www.st.cs.ru.nl/Onderzoek/Publicaties/publicaties.html

4.3.6 Hat

Report by:	Olaf Chitil
Participants:	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. 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 interoperate and use a similar command syntax.

Hat can be used both with nhc98 (→2.2) 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 year only small bug fixes were committed to the Darcs repository, but several other updates are also planned for the near future, including new and improved trace-browsers. A recent student project completed a Java-GUI viewer for traces, based on the idea of timelines and search. We hope this can be added to the repository soon.

Further reading

- http://www.haskell.org/hat
- `darcs get` http://darcs.haskell.org/hat
- Tracing and Debugging Functional Programs: http://www.cs.kent.ac.uk/~oc/tracing.html

4.3.7 Concurrent Haskell Debugger

Report by:	Fabian Reck
Participants:	Frank Huch, Jan Christiansen
Status:	experimental

Programming concurrent systems is difficult and error prone. The Concurrent Haskell Debugger is a tool for debugging and visualizing Concurrent Haskell and STM programs. By simply importing CHD.Control.Concurrent instead of Control.Concurrent and CHD.Control.Concurrent.STM instead of Control.Concurrent.STM the forked threads and their concurrent actions are visualized by a GUI. Furthermore, when a thread performs a concurrent action like writing an MVar or committing a transaction, it is stopped until the user grants permission. This way the user is able to determine the order of execution of concurrent actions. Apart from that, the program behaves exactly like the original program.

An extension of the debugger can automatically search for deadlocks and uncaught exceptions in the background. The user is interactively led to a program state where a deadlock or an exception was encountered. To use this feature, it is necessary to use a simple preprocessor that comes with the package that is available at http://www.informatik.uni-kiel.de/~fre/chd/.

Another purpose of the preprocessor is to enrich the source code with information for highlighting the next concurrent action in a source code view.

Future plans

- provide a more powerful preprocessor that is able to process imported modules
- add new views, like a visualization as a message sequence chart

- allow to undo concurrent actions

Further reading

- http://www.informatik.uni-kiel.de/~fre/docs/thesis.pdf (German diploma thesis)
- http://www.informatik.uni-kiel.de/~jac/data/ICFP2004.pdf

4.3.8 Hpc

Report by:	Andy Gill
Participants:	Colin Runciman
Status:	released and used

Haskell Program Coverage (HPC) is a set of tools for understanding program coverage. It consists of a source-to-source translator, an option (`-fhpc`) in ghc, a stand alone post-processor (`hpc`), a post-processor for reporting coverage, and an AJAX based interactive coverage viewer.

Hpc has been remarkably stable over the lifetime of ghc-6.8, with only a couple of minor bug fixes, including better support for .hsc files. The source-to-source translator is not under active development, and is looking for a new home. The interactive coverage viewer, which was under active development in 2007 at Galois, has now been resurrected at Hpc's new home in Kansas. Thank you Galois, for letting this work be released. The plan is to take the interactive viewer, and merge it with GHCi's debugging API, giving an AJAX based debugging tool.

Contact

⟨andygill@ku.edu⟩

4.3.9 SourceGraph

Report by:	Ivan Lazar Miljenovic
Status:	Version 0.4

SourceGraph is a utility program aimed at helping Haskell programmers visualize their code and perform simple graph-based analysis (representing functions as nodes in the graphs and function calls as directed edges). To do so, it utilizes the Graphalyze library (→ 5.8.4), which is designed as a general-purpose graph-theoretic analysis library. These two pieces of software are the focus of Ivan's mathematical honors thesis, "Graph-Theoretic Analysis of the Relationships Within Discrete Data", and are both available from Hackage.

Whilst fully usable, SourceGraph is currently limited in terms of input and output. It takes in the Cabal file of the project, and then analyzes all .hs and .lhs files recursively found in that directory. It utilizes Haskell-Src with Extensions, and should thus parse all extensions (with the current exception of Template Haskell, HaRP and HSX); files requiring C Pre-Processing are as yet unparseable, though this should be fixed in a future release. However, all functions defined in Class declarations and records are ignored due to difficulty in determining which actual instance is meant. The final report is then created in Html format in a "SourceGraph" subdirectory of the project's root directory.

Current analysis algorithms utilized include: alternative module groupings, whether a module should be split up, root analysis, clique and cycle detection as well as finding functions which can safely be compressed down to a single function. Please note however that SourceGraph is *not* a refactoring utility, and that its analyses should be taken with a grain of salt: for example, it might recommend that you split up a module, because there are several distinct groupings of functions, when that module contains common utility functions that are placed together to form a library module (e.g., the Prelude).

Further reading

- http://hackage.haskell.org/cgi-bin/hackage-scripts/package/SourceGraph
- http://code.haskell.org/SourceGraph

4.4 Development

4.4.1 Hoogle — Haskell API Search

Report by:	Neil Mitchell
Status:	v4.0

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 available as a web interface, a command line tool, and a lambdabot (→ 6.12.2) plugin.

Development of Hoogle was sponsored as part of the Google Summer of Code this year. As a result, a new version of Hoogle has been released with substantial speed and accuracy improvements. The next task is to generate Hoogle search information for all the libraries on Hackage.

Further reading

http://haskell.org/hoogle

4.4.2 Leksah, Haskell IDE

Report by:	Jürgen Nicklisch-Franken
Status:	in development

Leksah is a Haskell IDE written in Haskell based on Gtk+ and gtk2hs (→5.11.1). Leksah is a practical tool to support the Haskell development process. It is platform independent and should run on any platform where GTK+, gtk2hs, and GHC can be installed. (It is currently being tested on Windows and Linux but it should work on the Mac. It only works with GHC.)

There are compelling reasons for a Haskell IDE written in Haskell. First and most importantly, Haskell is different from mainstream imperative and object oriented languages and a dedicated IDE may exploit this specialness. Second the integration with an existing tool written in a different language has to solve the problem of integration of different programming languages/paradigms.

Currently Leksah offers features like jumping to definition for a name, integration of Cabal (→5.1) for building, Haskell source editor with "source candy", configurable keymaps, . . . This list will (hopefully) expand quickly.

The development of Leksah started in June 2007 and the first alpha version was released February 2008. Contributions of all kind are welcome.

Further reading

http://leksah.org/

4.4.3 EclipseFP — Haskell support for the Eclipse IDE

Report by:	Leif Frenzel
Status:	alpha

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

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

Over the past year, a new subproject called Cohatoe has developed a framework that allows us to implement Eclipse Plugins partly in Haskell. We are currently re-implementing and extending EclipseFP functionality in Haskell, using libraries such as Cabal (→5.1) and the GHC API (→2.1).

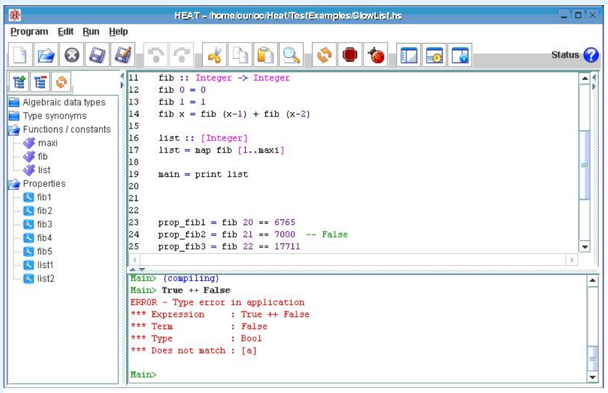
Further reading

- http://eclipsefp.sf.net
- http://leiffrenzel.de/eclipse/wiki/
- http://lists.sourceforge.net/lists/listinfo/eclipsefp-develop

4.4.4 HEAT: The Haskell Educational Advancement Tool

Report by:	Olaf Chitil

Heat is an interactive development environment (IDE) for learning and teaching Haskell. Using a separate editor and interpreter provides many distracting obstacles for inexperienced students learning a programming language; for example, they have to keep the versions in editor, interpreter, and in the file system in sync. Professional interactive development environments, however, confuse and distract with their excessive features. Hence Heat was designed for novice students learning the functional programming language Haskell. Based on teaching experience, Heat provides a small number of supporting features and is easy to use. Heat is portable, small, and works on top of the Haskell interpreter Hugs.



```
fib :: Integer -> Integer
fib 0 = 0
fib 1 = 1
fib x = fib (x-1) + fib (x-2)

list :: [Integer]
list = map fib [1..maxi]

main = print list



prop_fib1 = fib 20 == 6765
prop_fib2 = fib 21 == 7000  -- False
prop_fib3 = fib 22 == 17711
```

```
Main> (compiling)
Main> True ++ False
ERROR - Type error in application
*** Expression     : True ++ False
*** Term           : False
*** Type           : Bool
*** Does not match : [a]

Main>
```

Heat provides the following features:

- Editor for a single module with syntax-highlighting and matching brackets.
- Shows the status of compilation: non-compiled; compiled with or without error.
- Interpreter console that highlights the prompt and error messages.
- If compilation yields an error, then the source line is highlighted and additional error explanations are provided.
- Shows a program summary in a tree structure, giving definitions of types and types of functions . . .
- Automatic checking of all (Boolean) properties of a program; results shown in summary.

Heat is implemented in Java. The older version 1.1 which has been used in teaching functional programming at the University of Kent since 2006 is available from the webpage. Version 3.0 will be published very soon.

Further reading

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

4.4.5 Haskell Mode Plugins for Vim

Report by:	Claus Reinke
Participants:	Haskell & Vim users
Status:	maintenance mode

The Haskell mode plugins for Vim offer some IDE-style functionality for editing Haskell code in Vim, including several *insert mode completions* (based on identifiers currently in scope, on identifiers documented in the central Haddock indices, on tag files, or on words appearing in current and imported sources), *quickfix mode* (call compiler, list errors in quickfix window, jump to error locations in source window), *inferred type tooltips* (persisted from last successful `:make`, so you can still see some types after introducing errors, or use types to guide editing, e.g., function parameter order), various *editing helpers* (insert import statement, type declaration or module qualifier for id under cursor, expand implicit into explicit import statement, add option and language pragmas, . . .), and *direct access to the Haddock documentation* for the id under cursor.

The plugins are distributed as a simple vimball archive, including help file (after installation, try `:help haskellmode`). They derive their functionality and semantic information from GHC/GHCi, from Haddock-generated documentation and indices (→ 4.2.1), and from Vim's own configurable program editing support. For details of Haskell mode features, see the `haskellmode.txt` help file, for change log, see `haskellmode-files.txt` (for more general information, see Vim's excellent built-in `:help`, or browse the help files online at http://vimdoc.sourceforge.net/htmldoc/usr_toc.html).

These are not the only Haskell-related plugins for Vim — please add your own tricks and tips at haskell.org (syntax-coloring works out of the box, other scripts deal with indentation, . . .).

The haskellmode plugins for Vim are currently in maintenance mode, with infrequent updates and bug fixes, and the occasional new feature or improvement of older code (please let me know if anything does not work as advertised!).

Further reading

◦ Haskell Mode Plugins for Vim: http://www.cs.kent.ac.uk/~cr3/toolbox/haskell/Vim/
◦ haskell.org section listing these and other Vim files: http://www.haskell.org/haskellwiki/Libraries_and_tools/Program_development#Vim

4.4.6 yi

Report by:	Jean-Philippe Bernardy
Participants:	Don Stewart
Status:	active development

Yi is an editor written in Haskell and extensible in Haskell. We leverage the expressiveness of Haskell to provide an editor which is powerful and easy to extend.

We have recently made Yi much more accessible. On Unix systems, it can be deployed using a single **cabal install** command. We also polished the user interface and behavior to facilitate the transition from emacs or vim.

Yi features:

◦ Key-bindings for emacs and vim, written as extensible parsers;

◦ Vty, Gtk2Hs frontends;

◦ Syntax highlighting for Haskell and other languages;

◦ XMonad-style static configuration;

◦ Support of Linux, MacOS, and Windows platforms.

◦ Special support for Haskell: layout-aware edition, paren-matching, GHCi interface, Cabal build interface, . . .

We are currently working on the following fronts:

◦ Integration with Cabal and GHC API;

◦ Pango and Cocoa frontends;

◦ CUA key-bindings

Further reading

◦ More information can be found at: http://haskell.org/haskellwiki/Yi
◦ The source repository is available: `darcs get` http://code.haskell.org/yi/

4.4.7 HaRe — The Haskell Refactorer

Report by:	Huiqing Li
Participants:	Chris Brown, Chaddaï Fouché, Claus Reinke, Simon Thompson

Refactorings are source-to-source program transformations which change program structure and organization, but not program functionality. Documented in catalogs 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, generalization 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.

In order to allow users to extend HaRe themselves, HaRe includes an API for users to define their own program transformations, together with Haddock (→ 4.2.1) documentation. Please let us know if you are using the API.

HaRe 0.4, which was released earlier this year, only compiles with GHC 6.6.1 and GHC 6.8.2. New refactorings in HaRe 0.4 include a suite of slicing utilities, adding/removing a data constructor, converting a data type into a newtype, adding/removing a field, etc.

Snapshots of HaRe are available from our webpage, as are related presentations and publications from the group (including LDTA'05, TFP'05, SCAM'06, PEPM'08, and Huiqing's PhD thesis). The final report for the project appears there, too.

Recent developments

- More structural and datatype-based refactorings have been studied by Chris Brown, including transformation between `let` and `where`, generative folding, introducing pattern matching, and introducing case expressions;
- Chris has also been looking into duplicated code detection and elimination support for Haskell programs as part of his PhD research;
- Chaddaï Fouché started to work on the porting of HaRe to GHC API (→ 2.1) during the summer; this work is ongoing.

Further reading

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

4.4.8 DarcsWatch

Report by:	Joachim Breitner
Status:	working

DarcsWatch is a tool to track the state of Darcs (→ 6.1.1) patches that have been submitted to some project, usually by using the `darcs send` command. It allows both submitters and project maintainers to get an overview of patches that have been submitted but not yet applied.

During the last six months, development activity has been low, but DarcsWatch has been successfully used in the xmonad project (→ 6.1.2) to review unapplied and possibly forgotten patches before a release. Still, a good way to easily mark patches as obsolete or retracted has to be found.

Further reading

- http://darcswatch.nomeata.de/
- http://darcs.nomeata.de/darcswatch/documentation.html

4.4.9 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 when no C compiler is available (e.g., on Windows); and it understands the lexical syntax of Haskell, so you do not 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.

The current release is 1.6: recent bugfixes have been small — the major changes are to add new command-line options `-include` and `-strip-eol`.

Further reading

http://haskell.org/cpphs

5 Libraries

5.1 Cabal and Hackage

Report by: Duncan Coutts

Background

Cabal is the Common Architecture for Building Applications and Libraries. It defines a common interface for defining and building Haskell packages. It is implemented as a Haskell library and associated tools which allow developers to easily build and distribute packages.

Hackage is an online database of Cabal packages which can be interactively queried via the website and client-side software such as cabal-install. Hackage enables end-users to download and install Cabal packages.

cabal-install is the command line interface for the Cabal and Hackage system. It provides a command line program `cabal` which has sub-commands for installing and managing Haskell packages.

Recent progress

Since the last HCAR we have released Cabal-1.4 and 1.6. There have also been releases of cabal-install which is now at version 0.6.

The Cabal-1.4 and 1.6 releases contained a number of incremental improvements but nothing earth-shattering. Cabal-1.4 contained the bulk of the improvements but remained compatible with Cabal-1.2. The 1.6 release comes with GHC 6.10 and contains some new features that are not backwards compatible.

The cabal-install tool has matured significantly since the last report and is now mostly usable for many users. It replaces `runhaskell Setup.hs` which had been the primary interface for most users previously. The major advantage is that it simplifies the process of downloading and installing collections of inter-dependent packages from Hackage.

Hackage is getting near to its second birthday. At the time of writing, 234 users have uploaded 2306 versions of 820 packages. This represents a substantial amount of Haskell code and indeed a substantial amount of code re-use.

In the next few months we expect to upgrade Hackage with a new implementation. There should be no disruption for users or package maintainers. The new implementation uses HAppS rather than Apache+CGI. The main reason for the change is to make it easier to add the new features that users have been asking for. We also expect it will have a new layout and appearance thanks to the talented people from Tupil (→ 7.8). We are also hoping to use Hoogle (→ 4.4.1) as the primary search interface.

Google Summer of Code projects

Andrea Vezzosi completed his project to build a "make-like" dependency framework for the Cabal library. Part of this can now be integrated to make Cabal work better with pre-processors. To make development easier, Andrea started on an external `hbuild` tool. The aim is to improve it, demonstrate it on real projects, and eventually replace much of the Cabal internals. One remaining challenge is to develop some high level combinators for the dependency infrastructure.

Neil Mitchell completed his project on Hoogle 4 (→ 4.4.1). As mentioned above, the next step is to use it in the new Hackage server.

Looking forward

There is huge potential for Hackage to help us manage and improve the community's package collection. cabal-install is now able to report build results and the new Hackage server implementation can accept them. This should provide us with a huge amount of data on which packages work in which environments and configurations. More generally there is the opportunity to collect all sorts of useful metrics on the quality of packages.

As for Cabal, it now has the feature-set that enables it to support the vast majority of simple packages. The next challenge is large existing projects which have more complex requirements for a configuration and build system. Now may be the time to take a step back and discuss a new design document for Cabal 2.0.

People

We would like to thank the people who contributed to the last round of development work. Thanks also to the people who have followed development and reported bugs and feature requests.

We have ambitious plans and there is plenty of work to do, but not quite enough volunteer time to do it all. Now would be an excellent time to get involved in this central piece of the Haskell infrastructure.

Further reading

∘ Cabal homepage: http://www.haskell.org/cabal

- Hackage package collection: http://hackage.haskell.org/
- Bug tracker: http://hackage.haskell.org/trac/hackage/

5.2 Haskell Platform

Report by:	Duncan Coutts

Background

The Haskell Platform (HP) is the name of a new "blessed" set of libraries and tools on which to build further Haskell libraries and applications. It takes the best packages from the over 800 on Hackage (→ 5.1). It is intended to provide a comprehensive, stable, and quality tested base for Haskell projects to work from.

Historically, GHC has shipped with a collection of packages under the name `extralibs`. The intention in future is for GHC to get out of the business of shipping an entire platform and for this role to be transferred to the Haskell Platform.

Looking forward

We expect the first release of the platform to come out a few weeks after the release of GHC-6.10.1 (→ 2.1). Subsequent releases will be on a 6 month schedule. The first release will contain just the packages from the old `extralibs` collection, plus `cabal-install` (→ 5.1).

We are looking for involvement from the community to decide what procedures we should use and what level of quality we should demand for new additions to the platform. The discussion will take place on the `libraries@haskell.org` mailing list.

Further reading

http://haskell.org/haskellwiki/Haskell_Platform

5.3 Auxiliary Libraries

5.3.1 libmpd

Report by:	Ben Sinclair
Participants:	Joachim Fasting
Status:	maintained

LIBMPD is a client implementation of the MPD music playing daemon's network protocol. The interface has mostly stabilized and is usable. In version 0.3.1 some bugs have been addressed to fix the automatic reconnection feature and to be more permissive with data from the server.

Further reading

The development web page is at http://turing.une.edu.au/~bsinclai/code/libmpd-haskell/ and MPD can be found at http://www.musicpd.org/.

5.3.2 gravatar

Report by:	Don Stewart
Status:	active development

Gravatars (http://gravatar.com) are globally unique images associated with an email address, widely used in social networking sites. This library lets you find the URL of a gravatar image associated with an email address.

Further reading

- Source and documentation can be found on Hackage.
- The source repository is available:
  `darcs get` http://code.haskell.org/~dons/code/gravatar/

5.3.3 mersenne-random

Report by:	Don Stewart
Status:	active development

The Mersenne twister is a pseudorandom number generator developed by Makoto Matsumoto and Takuji Nishimura that is based on a matrix linear recurrence over a finite binary field. It provides for fast generation of very high quality pseudorandom numbers.

This library uses SFMT, the SIMD-oriented Fast Mersenne Twister, a variant of Mersenne Twister that is much faster than the original. It is designed to be fast when it runs on 128-bit SIMD. It can be compiled with either SSE2 OR PowerPC AltiVec support, to take advantage of these instructions.

By default the period of the function is $2^{19937} - 1$, however, you can compile in other defaults. Note that this algorithm on its own is not cryptographically secure.

Further reading

- Source and documentation can be found on Hackage.
- The source repository is available:
  `darcs get` http://code.haskell.org/~dons/code/mersenne-random/

5.3.4 cmath

Report by:	Don Stewart
Status:	active development

cmath is a complete, efficient binding to the standard C math.h library, for Haskell.

Further reading

- Source and documentation can be found on Hackage.

◦ The source repository is available:
`darcs get` http://code.haskell.org/~dons/code/cmath/

5.3.5 hmatrix

Report by:	Alberto Ruiz
Status:	stable, maintained

This library provides a purely functional interface to linear algebra and other numerical computations, internally implemented using GSL, BLAS, and LAPACK. The most common matrix computations are already available: eig, svd, chol, qr, hess, schur, lu, pinv, expm, etc. There are also functions for numeric integration and differentiation, nonlinear minimization, polynomial root finding, and many GSL special functions. The latest stable version can be found on Hackage.

Recent developments include low level optimizations contributed by Don Stewart and safe in-place updates using the ST monad.

Further reading

http://www.hmatrix.googlepages.com

5.3.6 The Neon Library

Report by:	Jurriaan Hage

As part of his master thesis work, Peter van Keeken implemented a library to data mine logged Helium (→2.4) programs to investigate aspects of how students program Haskell, how they learn to program, and how good Helium is in generating understandable feedback and hints. The software can be downloaded from http://www.cs.uu.nl/wiki/bin/view/Hage/Neon, which also gives some examples of output generated by the system. The downloads only contain a small sample of loggings, but it will allow programmers to play with it.

The recent news is that a paper about Neon has been (conditionally) accepted at SLE (1st Conference on Software Language Engineering, mainly people into UML, Java, and modeling can be found there), where it came under the heading of Tools for Language Usage. I have been considering to write a research proposal for actually performing some of these analyses (which I hope to get done before January), and with a publication under the belt this may actually work. I am still looking for know-how in the empirical arena.

5.3.7 unamb

Report by:	Conal Elliott
Status:	active development

Unamb is a package containing the *unambiguous choice* operator `unamb`, which wraps thread racing up in a purely functional, semantically simple wrapper. Given any two arguments `u` and `v` that agree unless bottom, the value of `unamb u v` is the more terminating of `u` and `v`. Operationally, the value of `unamb u v` becomes available when the earlier of `u` and `v` does. The agreement precondition ensures unamb's referential transparency.

Further reading

http://haskell.org/haskellwiki/unamb

5.4 Processing Haskell

5.4.1 hint

Report by:	Daniel Gorin
Status:	active
Current release:	0.2.5

This library defines a Haskell Interpreter monad. It allows to load Haskell modules, browse them, type-check and evaluate strings with Haskell expressions, and even coerce them into values. The operations are thread-safe and type-safe (even the coercion of expressions to values).

It may be useful for those who need GHCi-like functionality in their programs but do not want to mess with the GHC-API innards. Additionally, unlike the latter, hint provides an API that is consistent across GHC versions.

Works with GHC 6.6.x and 6.8.x. Upcoming version 0.3.0.0 will also work with GHC 6.10

Further reading

The latest stable version can be downloaded from Hackage.

5.4.2 mueval

Report by:	Gwern Branwen
Participants:	Andrea Vezzosi, Daniel Gorin, Spencer Janssen
Status:	active development

Mueval is a code evaluator for Haskell; it employs the GHC API as provided by the Hint library (→5.4.1). It uses a variety of techniques to evaluate arbitrary Haskell expressions safely & securely. Since it was begun in June 2008, tremendous progress has been made; it is currently used in Lambdabot (→6.12.2) live in #haskell). Mueval can also be called from the command-line.

Mueval features:

◦ A comprehensive test-suite of expressions which should and should not work

- Defeats all known attacks
- Optional resource limits and module imports
- The ability to load in definitions from a specified file
- Parses Haskell expressions with haskell-src-exts and tests against black- and white-lists
- Cabalized

We are currently working on the following:

- Refactoring modules to render Mueval more useful as a library
- Removing the POSIX-only requirement

Further reading

The source repository is available: `darcs get` http://code.haskell.org/mubot/

5.4.3 hscolour

Report by:	Malcolm Wallace
Status:	stable, maintained

HsColour is a small command-line tool (and Haskell library) that syntax-colorizes 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), LaTeX, and IRC chat codes. In all cases, the colors 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 (→ 4.2.1) 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. The current version is 1.10, which simply improves the title element on HTML output.

Further reading

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

5.5 Parsing and Transforming

5.5.1 pcre-light

Report by:	Don Stewart
Status:	active development

A small, efficient, and portable regex library for Perl 5 compatible regular expressions. The PCRE library is a set of functions that implement regular expression pattern matching using the same syntax and semantics as Perl 5.

Further reading

- Source and documentation can be found on Hackage.
- The source repository is available: `darcs get` http://code.haskell.org/~dons/code/pcre-light/

5.5.2 HStringTemplate

Report by:	Sterling Clover

HStringTemplate is a port of the StringTemplate library to Haskell. StringTemplate is a templating system that enforces strict model-view separation via a Turing-incomplete grammar that nonetheless provides powerful recursive constructs. The library provides template grouping and inheritance, as well as escaping. It is especially suited for rapid and iterative development of web applications. In the last period, a series of minor bugs in options handling have been resolved, but the code is otherwise stable and finding occasional use. HStringTemplate is currently at release 0.4 and is available via Hackage.

Further reading

- http://www.cs.usfca.edu/~parrt/papers/mvc.templates.pdf
- HStringTemplate: http://fmapfixreturn.wordpress.com
- StringTemplate: http://www.stringtemplate.org/

5.5.3 CoreErlang

Report by:	Henrique Ferreiro García
Participants:	David Castro Pérez
Status:	Parses and pretty-prints all of Core Erlang

CoreErlang is a Haskell library which consists of a parser and pretty-printer for the intermediate language used by Erlang. The parser uses the Parsec library, and the pretty-printer was modeled after the corresponding module of the haskell-src package. It also exposes a Syntax module which allows easy manipulation of terms.

It is able to parse and pretty-print all of Core Erlang. Remaining work includes customizing the pretty printer and refining the syntax interface.

Further reading

- It can be downloaded from hackage
- A darcs repository is available at: http://code.haskell.org/CoreErlang

5.5.4 parse-dimacs: A DIMACS CNF Parser

Report by:	Denis Bueno
Status:	Version 1.1

Parse-dimacs is a Parsec parser for a common file format — DIMACS — describing conjunctive normal form (CNF) formulas. CNF formulas are typically used as input to satisfiability solvers.

The parser is available from Hackage: http://hackage.haskell.org/cgi-bin/hackage-scripts/package/parse-dimacs

The next release will concentrate on optimization, specifically for large CNF formulas. The interface is simple and should be stable.

5.5.5 The X-SAIGA Project

Report by:	Richard A. Frost
Participants:	Rahmatullah Hafiz, Paul Callaghan
Status:	code available

The goal of the X-SAIGA project is to create algorithms and implementations which enable language processors (recognizers, parsers, interpreters, translators, etc.) to be constructed as modular and efficient embedded eXecutable SpecificAtIons of GrAmmars.

To achieve modularity, we have chosen to base our algorithms on top-down parsing. To accommodate ambiguity, we implement inclusive choice through backtracking search. To achieve polynomial complexity, we use memoization. We have developed an algorithm which accommodates direct left-recursion using curtailment of search. Indirect left recursion is also accommodated using curtailment together with a test to determine whether previously computed and memoized results may be reused depending on the context in which they were created and the context in which they are being considered for reuse.

The algorithm is described more fully in Frost, R., Hafiz, R., and Callaghan, P. (2007) Modular and Efficient Top-Down Parsing for Ambiguous Left-Recursive Grammars. Proceedings of the 10th International Workshop on Parsing Technologies (IWPT), ACL-SIGPARSE. Pages: 109–120, June 2007, Prague. (http://cs.uwindsor.ca/~hafiz/iwpt-07.pdf)

We have implemented our algorithms, at various stages of their development, in Miranda (up to 2006) and in Haskell (from 2006 onwards). A description of a Haskell implementation of our 2007 algorithm can be found in Frost, R., Hafiz, R., and Callaghan, P. (2008) Parser Combinators for Ambiguous Left-Recursive Grammars. Proceedings of the 10th International Symposium on Practical Aspects of Declarative Languages (PADL), Paul Hudak, David Scott Warren (Eds.): Practical Aspects of Declarative Languages, 10th International Symposium, PADL 2008, San Francisco, CA, USA, January 7–8, 2008. Springer 2008, LNCS 4902, 167–181. (http://cs.uwindsor.ca/~hafiz/PADL_PAPER_FINAL.pdf)

The X-SAIGA website contains more information, links to other publications, proofs of termination and complexity, and Haskell code of the development version. (http://cs.uwindsor.ca/~hafiz/proHome.html)

We are currently extending our algorithm and implementation to accommodate executable specifications of fully-general attribute grammars.

One of our long-term goals is to use the X-SAIGA software to construct natural-language applications as executable specifications of attribute grammars and deploy them on the Public-Domain SpeechWeb, which is a related project of ours that is also funded by the Natural Science and Engineering Research Council of Canada (NSERC). More information on the SpeechWeb project, including details of how to access our prototype Public-Domain SpeechWeb by voice, and how to build and deploy your own speech applications, can be found at http://www.myspeechweb.org.

5.5.6 InterpreterLib

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

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 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 re-use. The library also implements a specialization of the SmashA (→ 1.5.6) generic programming technique to support generic default algebras and to override those defaults with functor-specific behavior. 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, re-use, and simplified maintenance of language processors.

The Oread (→ 6.9.4) implementation employs InterpreterLib.

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⟩

5.5.7 KURE

Report by:	Andy Gill
Status:	alpha

The Kansas University Rewrite Engine (KURE, pronounced cure) is a DSL for writing rewrite systems over grammars with scope. It was used (along with Template Haskell) to provide the basic rewrite abilities inside HERA (Haskell Equational Reasoning Assistant). It has been recently rewritten and will be published on hackage shortly. KURE provides combinators for ordering the application of an abstract type, `Rewrite`, combinators for building primitive rewrites, and combinators performing rewrite searches. We plan to use KURE to explore some rewrites inside our low-level hardware description language Oread (→ 6.9.4), as well as power the next version of HERA.

Contact

⟨andygill@ku.edu⟩

5.5.8 Typed Transformations of Typed Abstract Syntax (TTTAS)

Report by:	Arthur Baars
Participants:	Doaitse Swierstra, Marcos Viera
Status:	actively developed

The TTTAS library, which has an arrow like interface, supports the construction of analyses and transformations in a typed setting. The library uses *typed abstract syntax* to represent fragments of embedded programs containing variables and binding structures, while preserving the idea that the type system of the host language is used to emulate the type system of the embedded language. Internally the library maintains a collection of binding structures of the EDSL. A transformation may introduce new bindings, and the binding may even be mutually recursive. The library ensures that in the end the bindings resulting from a transformation are consistent.

Introduction

Advantages of embedded domain-specific languages (EDSLs) are that one does not have to implement a separate type system nor an abstraction mechanism, since these are directly borrowed from the host language. Straightforward implementations of embedded domain-specific languages map the semantics of the embedded language onto a function in the host language. The semantic mappings are usually compositional, i.e., they directly follow the syntax of the embedded language.

One of the questions which arises is whether conventional compilation techniques, such as global analysis and resulting transformations, can be applied in the context of EDSLs.

Run-time transformations on the embedded language can have a huge effect on performance. In previous work we present a case study comparing the `Read` instances generated by Haskells `deriving` construct with instances on which run-time grammar transformations (precedence resolution, left-factorization and left-corner transformation) have been applied.

Background

The approach taken in TTTAS was proposed by Arthur Baars, Doaitse Swierstra, and Marcos Viera.

The library is employed to implement the transformations used in the Haskell 2008 paper "Haskell, Do You Read Me? Constructing and Composing Efficient Top-down Parsers at Runtime" (→ 5.5.9).

Future plans

A first version of TTTAS will soon be released on Hackage.

Further reading

More information can be found on the TTTAS home page.

5.5.9 Grammar Based Read (GRead)

Report by:	Marcos Viera
Participants:	Doaitse Swierstra, Eelco Lempsink
Status:	actively developed

The GRead library provides an alternative to the standard `read` function. Instead of composing parsers, we dynamically compose grammars describing datatypes, removing any left-recursion and combining common prefixes of alternatives. From the composed grammar, we generate a final parser using a function `gread` that has a few improvements over `read`.

Introduction

The Haskell definition and implementation of `read` is far from perfect. First, `read` is not able to handle the infix operator associativity. This also puts constraints on the way `show` is defined and forces it to generate more parentheses than necessary. Second, it may give rise to exponential parsing times. These problems are due to the compositionality requirement for `read` functions, which imposes a top-down parsing strategy. GRead provides a different approach, based on typed abstract syntax, in which grammars describing the datatypes are composed dynamically.

We define a function `gread` with the following features:

- Handle the associativity of infix operators. The corresponding `gshow` generates fewer parentheses than the standard `show`.
- Read data in linear time. The standard `read` has an exponential behavior in some cases of datatypes with infix operators.
- Is able to repair possible errors in the input.

The instances of the class `Gram` (that make grammar first-class values) can be automatically derived using the function `deriveGrammar`.

Background

The approach taken in GRead was proposed by Marcos Viera, Doaitse Swierstra, and Eelco Lempsink in the Haskell 2008 paper "Haskell, Do You Read Me? Constructing and Composing Efficient Top-down Parsers at Runtime." It uses the Typed Transformations of Typed Abstract Syntax library (→ 5.5.8) developed by Arthur Baars and Doaitse Swierstra.

Future plans

A first version of GRead will soon be released on Hackage.

Further reading

See the TTTAS home page.

5.5.10 Utrecht Parser Combinator Library

Report by:	Doaitse Swierstra
Status:	actively developed

The Utrecht Parser Combinator library has remained largely unmodified for the last five years, and has served us well. Over the years, however, new insights have grown, and with the advent of GADTs some internals could be simplified considerably. The Lernet summer school in February 2008 (http://www.fing.edu.uy/inco/eventos/lernet2008/) provided an incentive to start a rewrite of the library; a newly written tutorial will appear in the lecture notes.

Features

- Much simpler internals than the old library.
- Online result production, error recovery, combinators for parsing ambiguous grammars, an applicative interface, a monadic interface.
- Scanners can be switched dynamically, so several different languages can occur intertwined in a single input file.
- Fixes a potential black hole which went unnoticed for years in the code for the monadic bind as presented by Swierstra and Hughes in the ICFP 2003 paper: *Polish Parsers: Step by Step.*
- Surprisingly, and counter-intuitively, it has turned out that the abstract interpretation as described by Swierstra and Duponcheel in the lecture notes of the school on Advanced Functional programming in 1996 can be combined with monadic operations. However, this part has not been fully implemented yet in the new library.

Although the library has not been released yet, it has been used successfully in a couple of Utrecht-internal projects.

Future plans

The final library, with the abstract interpretation part in order to get the parsing speed we got used to, will be release on Hackage again. We plan to extend the short tutorial which will appear in the LNCS series (45 pages) into a long tutorial.

Contact

If you are interested in using the current version of the library in order to provide feedback on the provided interface, contact ⟨doaitse@swierstra.net⟩.

5.6 Mathematical Objects

5.6.1 dimensional

Report by:	Björn Buckwalter
Status:	active, mostly stable

Dimensional is a library providing data types for performing arithmetics 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.

The core of dimensional is stable with additional units being added on an as-needed basis. In addition to the SI system of units, dimensional has experimental support for user-defined dimensions and a proof-of-concept implementation of the CGS system of units. I am also experimenting with forward automatic differentiation and rudimentary linear algebra.

The current release is compatible with GHC 6.6.x and above and can be downloaded from Hackage or the project web site. The primary documentation is the literate Haskell source code, but the wiki on the project web site has a few usage examples to help with getting started.

The Darcs repo has moved to http://code.haskell.org/dimensional.

Further reading

http://dimensional.googlecode.com

5.6.2 Halculon: units and physical constants database

Report by:	Jared Updike
Status:	web application in beta, database stable

A number of Haskell libraries can represent numerical values with physical dimensions that are checked at runtime or compile time (including dimensional and the Numeric Prelude), but neither provide an exhaustive, searchable, annotated database of units, measures, and physical constants. Halculon is an interactive unit database of 4,250 units, with a sample Haskell AJAX web application, based on the units database created by Alan Eliasen for the wonderful physical units programming language Frink. (Because each unit in Frink's unit.txt database is defined in terms of more basic unit definitions — an elegant approach in general — units.txt is inconvenient for looking up a single random unit; the entire file might need to be parsed to represent any given constant solely in terms of the base SI units, which is precisely what the Halculon database provides.)

Halculon also provides a carefully tuned, user- and developer-friendly search string database that aims to make interactive use pleasant. The database tables are available online and downloadable as UTF-8 text.

Future plans for the sample calculator web application include utilizing MPFR's arbitrary precision floats to bring greater range to Real calculations, in line with those for Integers and Rationals (built in to Haskell).

Further reading

- http://www.updike.org/articles/Units
- http://www.updike.org/halculon/

5.6.3 Numeric prelude

Report by:	Henning Thielemann
Participants:	Dylan Thurston, 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 lazy Peano number, arbitrarily quantified non-negative lazy number (generalization of Peano number), complex number, quaternion, residue class, fraction, partial fraction, number equipped with physical units in two variants (dynamically and statically checked) fixed point number 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. The library has basic Cabal support and a growing test-suite of QuickCheck tests for the implemented mathematical objects.

Each data type now resides in a separate module. Cyclic dependencies could be eliminated by fixing some types in class methods. E.g., power exponents became simply Integer instead of Integral, which has also the advantage of reduced type defaulting.

Further reading

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

5.6.4 vector-space

Report by:	Conal Elliott
Status:	active development

`vector-space` is library that provides provides classes and generic operations for additive groups, vector spaces and affine spaces. There are also vector space bases and a general notion of linear maps. The library also defines a type of infinite towers of generalized derivatives. A generalized derivative is a linear map rather than one of the usual concrete representations (scalars, vectors, matrices, . . .).

For the past few months, this work has been graciously supported by Anygma.

Further reading

http://haskell.org/haskellwiki/vector-space

5.6.5 Nat

Report by:	Jan Christiansen
Status:	experimental

Nat is a small library that provides an implementation of natural numbers. It was motivated by a similar implementation in the functional logic programming language Curry (→3.2.1). In contrast to most other implementations it uses a binary representation instead of Peano numbers. Therefore, the performance of arithmetic operations is substantially better. Furthermore, the operations are implemented in a least strict way. That is, they do only evaluate their arguments as far as necessary. It turned out that the implementation of least strict functions is not at all as trivial as one would expect. This implementation emerged from motivating examples for a tool to check whether a function is least strict, called StrictCheck. The implementation is available via hackage at http://hackage.haskell.org/cgi-bin/hackage-scripts/package/nat.

Further reading

http://www-ps.informatik.uni-kiel.de/currywiki/fun/naturals

5.6.6 AERN-Real and friends

Report by:	Michal Konečný
Participants:	Amin Farjudian, Jan Duracz
Status:	experimental, actively developed

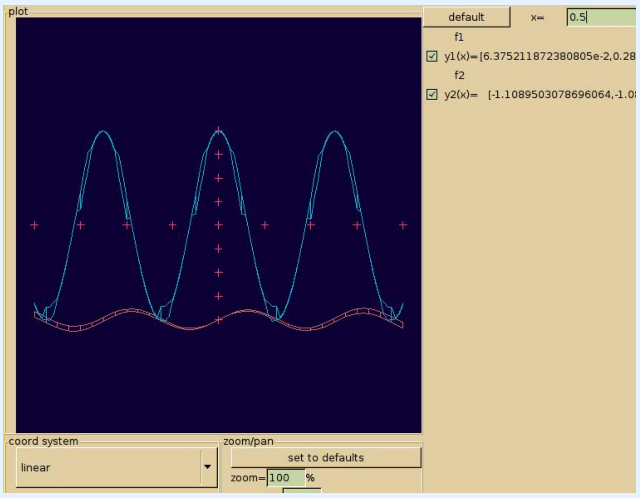
AERN stands for Approximating Exact Real Numbers. We are developing a family of the following libraries for fast exact real number arithmetic:

- AERN-Real: arbitrary precision interval arithmetic with multiple backends (pure Haskell floating point numbers, MPFR, correctly rounded doubles)

- AERN-RnToRm: arbitrary precision arithmetic of piece-wise polynomial function enclosures (PFEs) for functions over n-dimensional real intervals

- AERN-RnToRm-Plot: GTK window for inspecting the graphs of PFEs in one variable (see figure below, showing a screenshot of an AERN-RnToRm-Plot window exploring an enclosure of $cos(10x)$ (blue) and an enclosure of its primitive function (red))

- AERN-Net: an implementation of distributed query-based (i.e., lazy) computation over analytical and geometrical objects



x= 0.5

f1

☑ y1(x)=[6.375211872380805e-2,0.28

f2

☑ y2(x)= [-1.1089503078696064,-1.0

linear

zoom= 100 %

The development is driven mainly by the needs of our two research projects. We use the libraries extensively to:

- prototype algorithms for reliable and ultimately converging methods for solving differential equations in many variables (AERN-RnToRm, AERN-Net)

- solve numerical constraint satisfaction problems, especially those arising from verification of programs that use floating point numbers (AERN-RnToRm)

For our purposes AERN-Real has been stable for almost a year. It needs to be tested for a wider variety of applications before we can label it as stable. The other libraries are very likely to contain errors and we discover some every now and then. Also their API is occasionally changing. All the libraries provide a fairly extensive set of features and are reasonably well documented.

The libraries are under active development and new features and bug fixes are expected to be submitted to Hackage for at least the whole of 2008/2009. Notable planned additions in this period include:

- lazy communication of approximations of higher-order real functions using role switching

- infinite trees of enclosures for interval partial derivatives computed using automatic differentiation

- zooming, panning and better coordinate display in the GTK graph display

Further reading

See Haddock documentation via Hackage — has links to research papers.

5.6.7 Haskell BLAS Bindings

Report by:	Patrick O. Perry
Status:	version 0.6

The `blas` library is a set of high-level bindings to the Basic Linear Algebra Subprograms (BLAS). The project is part of a larger effort to make high performance scientific computing in Haskell possible.

The design goal is to make using BLAS as natural as possible without sacrificing functionality. In particular, the library has both pure and impure data types, the latter supporting destructive updates in the `ST` and `IO` monads.

The library has just undergone a massive rewrite to clean up the interface and support `ST` operations. At this point most of the core functionality is in place, but there may be some aesthetic changes in the future. The latest version is available on Hackage.

If anyone would like to contribute to the project, there is still plenty of work to do, and help is always appreciated. Work on bindings for the rest of LAPACK is about to begin.

Further reading

http://quantile95.com

5.7 Data types and data structures

5.7.1 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, 6.6, 6.8, Hugs 2005–2006, and the head version of nhc98.

Bytestring 0.9.1.0 has been released, with full coverage data, an improved testsuite, and some key performance improvements.

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://darcs.haskell.org/bytestring

5.7.2 bytestring-mmap

Report by:	Don Stewart
Status:	active development

This library provides a wrapper to mmap(2), allowing files or devices to be lazily loaded into memory as strict or lazy ByteStrings (→5.7.1), using the virtual memory subsystem to do on-demand loading.

Further reading

- Source and documentation can be found on Hackage.
- The source repository is available:
  `darcs get` http://code.haskell.org/~dons/code/bytestring-mmap/

5.7.3 dlist

Report by:	Don Stewart
Status:	active development

Differences lists: a list-like type supporting O(1) append. This is particularly useful for efficient logging and pretty printing, (e.g., with the Writer monad), where list append quickly becomes too expensive.

Further reading

- Source and documentation can be found on Hackage.
- The source repository is available:
  `darcs get` http://code.haskell.org/~dons/code/dlist/

5.7.4 HList — a library for typed heterogeneous collections

Report by:	Oleg Kiselyov
Participants:	Ralf Lämmel, Keean Schupke, Gwern Branwen

HList is a comprehensive, general purpose Haskell library for typed heterogeneous collections including extensible polymorphic records and variants (→1.5.6). HList is analogous to 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 compile-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), 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: http://darcs.haskell.org/HList

The main change since last year was the addition of a large set of patches by Gwern Branwen ⟨gwern0@gmail.com⟩, to arrange the library within the Data.HList hierarchy, to update the code for GHC 6.8.2 (using the LANGUAGE pragma, eliminating causes of GHC warnings), to build the library with the latest version of Cabal. He also uploaded the library to Hackage. Many thanks to Gwern Branwen.

Further reading

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

5.7.5 stream-fusion

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.

The latest version of the stream-fusion package is now available from Hackage.

Further reading

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

5.7.6 Edison

Report by:	Robert Dockins
Status:	stable, maintained

Edison is a library of purely function data structures for Haskell originally written by Chris Okasaki. Conceptually, it consists of two things:

1. A set of type classes defining data the following data structure abstractions: "sequences", "collections" and "associative collections"

2. Multiple concrete implementations of each of the abstractions.

The following major changes have been made since version 1.1, which was released in 1999.

- Typeclasses updated to use fundeps (by Andrew Bromage)
- Implementation of ternary search tries (by Andrew Bromage)
- Modules renamed to use the hierarchical module extension
- Documentation haddockized
- Source moved to a darcs repository
- Build system cabalized
- Unit tests integrated into a single driver program which exercises all the concrete implementations shipped with Edison
- Multiple additions to the APIs (mostly the associated collection API)

Edison is currently in maintain-only mode. I do not have the time required to enhance Edison in the ways I would like. If you are interested in working on Edison, do not hesitate to contact me.

The biggest thing that Edison needs is a benchmarking suite. Although Edison currently has an extensive unit test suite for testing correctness, and many of the data structures have proven time bounds, I have no way to evaluate or compare the quantitative performance of data structure implementations in a principled way. Unfortunately, benchmarking data structures in a non-strict language is difficult to do well. If you have an interest or experience in this area, your help would be very much appreciated.

Further reading

http://www.cs.princeton.edu/~rdockins/edison/home/

5.7.7 MemoTrie

Report by:	Conal Elliott
Status:	active development

MemoTrie is functional library for creating efficient memo functions, using tries. It is based on some code from Spencer Janssen and uses type families.

Further reading

http://haskell.org/haskellwiki/MemoTrie

5.8 Data processing

5.8.1 The Haskell Cryptographic Library

Report by:	Dominic Steinitz

The latest version is still 4.1.0. Contributions (in the form of, e.g., improvements to documentation) continue to be received.

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

This release contains:

- DES
- Blowfish
- AES
- TEA
- BubbleBabble
- Cipher Block Chaining (CBC)
- PKCS#5 and nulls padding
- SHA-1, SHA-2, SHA-224, SHA-256, SHA-384, SHA-512
- HMAC
- MD5
- RSA
- OAEP-based encryption (Bellare-Rogaway)
- Hex utilities
- Support for Word128, Word192 and Word256, and beyond

Further reading

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

5.8.2 The Haskell ASN.1 Library

Report by: Dominic Steinitz

The current release is 0.0.18, which contains functions to handle ASN.1, X.509, PKCS#8, and PKCS#1.5.

This still has a dependency on NewBinary but we now have a way of removing this by using ByteStrings (→ 5.7.1). The functions for handling 2s complement and non-negative binary integer encodings now use the new method. Unfortunately, Adam Langley has announced he is no longer able to support binary-strict which has been used to replace NewBinary. I have offered to take on its support but I have not yet got around to updating the hackage entry.

The current version handles the Basic Encoding Rules (BER). In addition, even more work (over 500 Darcs patches) has been undertaken on handling the Packed Encoding Rules (PER) using a GADT to represent the Abstract Syntax Tree (we will probably move the BER to use the same AST at some point). Interestingly, this has resulted in us finding a small bug in the ASN.1 specification which we have reported to the ITU.

I do not suggest downloading the current working version yet (unless you want to contribute). We are in the process of moving all the original tests across to work with the new version of the AST.

The plan is to write a formal and executable specification of ASN.1 PER as a technical report. Thereafter, we plan to release the documentation in haddock form.

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.

5.8.3 MultiSetRewrite

Report by: Martin Sulzmann

MultiSetRewrite is a Haskell library extension to support multi-set rewrite rules with guards. Rewrite rules are executed concurrently as long as they operate on non-overlapping left-hand sides. We make use of highly-concurrent, non-blocking data structures based on CAS lists. Thus, we can parallelize concurrent rule executions on a multi-core architecture. See for details http://sulzmann.blogspot.com/2008/10/multi-set-rewrite-rules-with-guards-and.html

Latest developments

The MultiSetRewrite implementation (as a library extension to Haskell) is available via http://hackage.haskell.org/cgi-bin/hackage-scripts/package/multisetrewrite

The package includes two example applications:

- A concurrent stack using Join-style concurrency implemented on top of MultiSetRewrite.
- A Constraint Handling Rules (CHR) solver.

5.8.4 Graphalyze

Report by: Ivan Lazar Miljenovic
Status: Version 0.3

The Graphalyze library is a general-purpose, fully extensible graph-theoretic analysis library, which includes functions to assist with graph creation and visualization, as well as many graph-related algorithms. Also included is a small abstract document representation, with a sample document generator utilizing Pandoc (→ 6.4.1). Users of this library are able to mix and match Graphalyze's algorithms with their own.

Graphalyze is used in SourceGraph (→ 4.3.9).

Further reading

- http://hackage.haskell.org/cgi-bin/hackage-scripts/package/Graphalyze
- http://code.haskell.org/Graphalyze

5.9 Generic Programming

5.9.1 uniplate

Report by:	Neil Mitchell

Uniplate is a boilerplate removal library, with similar goals to the original Scrap Your Boilerplate work. It requires fewer language extensions, and allows more succinct traversals with higher performance than SYB (→ 5.9.2). A paper including many examples was presented at the Haskell Workshop 2007. Since the original version, the library has been further optimized and is now about 15% faster.

If you are writing a compiler, or any program that operates over values with many constructors and nested types, you *should* be using a boilerplate removal library. This library provides a gentle introduction to the field, and can be used practically to achieve substantial savings in code size and maintainability.

Further reading

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

5.9.2 Scrap Your Boilerplate (SYB)

Report by:	José Pedro Magalhães
Participants:	Sean Leather
Status:	actively developed

Scrap Your Boilerplate (SYB) is a library for generic programming in Haskell. It has been supported by GHC since the 6.0 release. The library is based on combinators and a few primitives for type-safe casting and processing constructor applications.

It was originally developed by Ralf Lämmel and Simon Peyton Jones. Since then, many people have contributed with research relating to SYB or its applications.

Recent changes

In the discussion towards the release of the 6.10 version of GHC, it was decided that SYB would be separated from the compiler itself. This allows for easier maintainability, since updating the library does not have to depend on updating the compiler. This splitting amounts to moving the `Data.Generics` modules from the `base` package into a new package called `syb`.

One issue with splitting the `Data.Generics` modules is that the `Data` class is tightly coupled to GHC's automatic generation of instances. Completely moving the entire SYB library from the `base` package would give a false sense of separation, since the methods of `Data` cannot be changed without also modifying the compiler. As a result, `Data` was moved from the `Data.Generics.Basics` module to `Data.Data`. Discussion on how to split SYB resulted in this and other changes to the code. These changes not only allow the library to be developed independently from GHC but also reduce dependencies on SYB in cases where it is not necessary.

Future plans

The next step is to create a separate repository for the new `syb` package and develop it independently, releasing it on Hackage. There are several ideas for future improvements for SYB, namely increasing performance and providing more generic functions (such as generic map).

Contact

To report bugs or suggest improvements, please use the issue tracker for SYB. For general concerns and questions, please use the Generics mailing list.

Further reading

More information can be found on the new SYB home page. For API documentation, refer to the Haddock documentation. The original webpage also contains information and many examples.

5.9.3 Extensible and Modular Generics for the Masses (EMGM)

Report by:	Sean Leather
Participants:	José Pedro Magalhães, Alexey Rodriguez, Andres Löh
Status:	actively developed

Extensible and Modular Generics for the Masses (EMGM) is a general-purpose library for generic programming with type classes.

Introduction

EMGM embodies the concepts of datatype-generic programming in a library. Haskell datatypes are represented at the value level using a sum-of-products view. Generic functions are written by induction on the structure of the representation. The library also allows generic functions to be extended using ad-hoc cases for arbitrary datatypes. Adding support for new datatypes is straightforward.

Other features of the library include a sizable collection of ready-to-use generic functions and built-in support for standard datatypes. Some examples of the functions are:

- `Crush`, a useful generalization of fold-like operations that supports flattening, integer operations, and logic operations on all values of an arbitrary datatype
- Extensible `Read` and `Show` functions to which one might add special cases for certain types
- `Collect` for collecting values of a certain type contained within a value of a different type
- `ZipWith`, a generic version of the standard `zipWith`

EMGM also comes with support for standard datatypes such as lists, `Either`, `Maybe`, and tuples.

Background

The ideas for EMGM come from research by Ralf Hinze, Bruno Oliveira, and Andres Löh. It was further explored in a comparison of generic programming libraries by Alexey Rodriguez, et al. Our particular implementation was developed simultaneously with the writing of lecture notes for the 2008 Advanced Functional Programming Summer School.

Future plans

In the near future (possibly by the time you read this), we will release a new version of EMGM with support for generating the representation of a datatype using Template Haskell. Currently, this must be done manually. We also have many ideas about new generic functions and possible applications.

Contact

We are definitely interested in knowing if you use EMGM, how you use it, and where it can be improved. We may be contacted on the Generics mailing list.

Further reading

More information can be found on the EMGM home page. References for the research that resulted in EMGM can be found on the Hackage page for EMGM.

5.9.4 multirec: Generic programming with systems of recursive datatypes

Report by:	Alexey Rodriguez
Participants:	Stefan Holdermans, Andres Löh, Johan Jeuring
Status:	actively developed

Many generic programs require information about the recursive positions of a datatype. Examples include the generic fold, generic rewriting, or the Zipper data structure. Several generic programming systems allow to write such functions by viewing datatypes as fixed points of a pattern functor. Traditionally, this view has been limited to so-called regular datatypes such as lists and binary trees. In particular, systems of mutually recursive datatypes have been excluded.

With the multirec library, we provide a mechanism to talk about fixed points of systems of datatypes that may be mutually recursive. On top of this representations, generic functions such as the fold or the Zipper can then be defined.

We expect that the library will be especially interesting for compiler writers, because ASTs are typically systems of mutually recursive datatypes, and with multirec it becomes easy to write generic functions on ASTs.

Features and limitations

- Generalizes the fixed point view from single, regular datatypes to systems of recursive datatypes.
- Includes detailed examples: generic fold and generic compos (in the style of Bringert and Ranta, see below). The Zipper and generic rewriting for systems of datatypes will be released soon as separate libraries that build on multirec.
- The generic compos functions do not require the user to modify their existing systems of datatypes.
- In its current form, this library does not support nested datatypes.

Future plans

At the moment the user is required to enable rewriting on a datatype by supplying a type-specific instance declaration. In the future, we are planning to automate this process using Template Haskell.

Contact

Please do get in touch with us using the Generics mailing list (http://www.haskell.org/mailman/listinfo/generics) if you find the library useful or if you want to report bugs and make suggestions.

Further reading

- The library is available on Hackage as `multirec`. More information will be made available on the multirec home page (http://www.cs.uu.nl/wiki/GenericProgramming/Multirec).
- Paper about the ideas underlying the library: http://www.cs.uu.nl/~andres/Rec
- Paper about compos by Bringert and Ranta: http://www.cs.chalmers.se/~bringert/publ/composOp/composOp.pdf

5.9.5 Generic rewriting library for regular datatypes

Report by:	Alexey Rodriguez
Participants:	Thomas van Noort, Stefan Holdermans, Johan Jeuring, Bastiaan Heeren
Status:	actively developed

This library provides a set of functions to apply rewrite rules to a datatype. The rewrite functions are generic, so it is not necessary to re-implement the matching and substitution machinery for every datatype. Additionally, the library provides a set of generic traversal functions that can be used together with rewriting.

Features and limitations

- Generic rewriting machinery
- Generic traversals (top-down, bottom-up, etc.)
- Rewrite rules are just datatypes and therefore observable. This means that you can, for example, invert rewrite rules.
- Rewrite rules are defined in the original domain. So the user does not have to worry about internal implementation details. For instance, the generic extension of datatypes with metavariables remains internal to the library.
- This library can be used with regular datatypes, that is, datatypes that exhibit simple recursion such as lists and binary trees (nested datatypes and mutual recursion are not supported)

Future plans

At the moment the user is required to enable rewriting on a datatype by supplying a type-specific instance declaration. In the future, we are planning to automate this process using Template Haskell.

For a version of the library that supports mutual recursion, please have a look at *multirec* (→ 5.9.4).

Contact

Please do get in touch with us using the Generics mailing list if you find the library useful or if you want to report bugs and make suggestions.

Further reading

More information can be found on the Rewriting home page. References for the research that resulted in the rewriting library can be found on the Hackage page for rewriting.

5.9.6 2LT: Two-Level Transformation

Report by:	Tiago Miguel Laureano Alves
Participants:	Joost Visser, 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 output types.
- Front-ends for VDM-SL, XML, and SQL. These front-ends support (i) reading a schema, (ii) applying a two-level transformation system to produce a new schema, (iii) converting a document/database corresponding to the input schema to a document/database corresponding to the output schema, and *vice versa*.
- 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. Independently of two-level transformation, the combinators can be used to specialize structure-shy programs (such as XPath queries and strategic functions) to structure-sensitive point-free form, and *vice versa*.
- Support for schema constraints using point-free expressions. Constraints present in the initial schema are preserved during the transformation process and new constraints are added in specific transformations to ensure semantic preservation. Constraints can be simplified using the already existent library for transformation of point-free functions.

The various sets of transformation combinators are reminiscent of the combinators of Strafunski and the

Scrap-your-Boilerplate (→ 5.9.2) approach to generic functional programming.

A release of 2LT is available from the project URL.

Future plans

New functionality is planned, such as elaboration of the front-ends and the creation of a web interface. Furthermore, efforts are underway to reimplement the existent functionality using lenses under the context of the PhD student Hugo Pacheco.

Further reading

- Project URL: http://2lt.googlecode.com
- 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.* PEPM 2007.
- Tiago L. Alves, Paulo Silva and Joost Visser. *Constraint-aware Schema Transformation.* RULE, 2008.

5.10 Types for Safety and Reasoning

5.10.1 Takusen

Report by:	Alistair Bayley
Participants:	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 that 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 marshaling and unmarshaling 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. ODBC support is nearly complete; string output bind-variables do not marshal correctly.

Things have been quiet. Since the last report we have:

- added support for ODBC output bind variables (but this is not yet fully functional)
- improved support for Oracle output bind variables
- made many small fixes to the installation process

Future plans

- complete ODBC interface.
- Large object support.
- MS SQL Server and Sybase interfaces, via FreeTDS.

Further reading

- Hackage: http://hackage.haskell.org/cgi-bin/hackage-scripts/package/Takusen
- `darcs get` http://darcs.haskell.org/takusen/
- browse docs: http://darcs.haskell.org/takusen/doc/html (see Database.Enumerator for Usage instructions and examples)

5.10.2 Session Types for Haskell

Report by:	Matthew Sackman
Status:	beta; active development

Session Types provide a way to express communication protocols. They specify, for a bi-directional channel, who says what, in what order, and to whom. Looping and branching structures are supported. Thus a session type adds a type to a communication channel, ensuring that use of the channel is safe (i.e., when one party speaks, the others listen, and that the type of the value sent is the type of the value expected to be received). Thus, Session Types offer temporal information which is absent from all other concurrency techniques.

The focus of the library is on the communication between threads. However, work is progressing on supporting fully distributed operation. The library supports forking new processes with channels; creating new channels between existing processes; the communication of process IDs; the communication of channels (higher-order channels or delegation); subtyping of Pids; and some initial work on real distributed operation over Handles.

Current development is rapid and is focusing on building up a strong suite of examples and networked operation. Recent features have added support for higher-order channels and a new DSL for specifying Session Types (which supports lexical scoping and is composable).

If you are doing any multi-threaded development in Haskell and find the properties and simplicity of message passing concurrency attractive, then I strongly encourage you to take a look at Session Types.

Further reading

- Project homepage: http://wellquite.org/sessions/
- Tutorial:
  http://wellquite.org/sessions/tutorial_1.html

5.10.3 Category Extras — Comonad Transformers and Bird-Meertens combinators

Report by:	Edward Kmett
Status:	experimental

Haskell has derived a lot of benefits from the tools it has borrowed from category theory, such as functors, monads, and arrows. However, there are a lot more tools out there. For instance, comonads have been barely touched by Haskell programmers. This library attempts to collect many of the more interesting bits of category theory and constructive algorithmics in one place and generalize or dualize previous results where possible.

The library includes:

- A set of comonad transformers in the spirit of the monad transformer library, including the categorical duals of the Reader, State, and Writer monads and monad transformers: the Product, Context, and Supply comonads and comonad transformers, respectively.

- An expanded set of "Bananas, Lenses, and Barbed Wire" for constructive algorithmics, including new generalized hylomorphisms and combinators for building up the distributive laws needed to use them

- Left and right Kan extensions

- Generalizations of standard library functions such as zip and unzip

- Free monads and cofree comonads with free monad coproducts and cofree comonad coproducts

- Ideal monads and their previously unpublished dual, coideal comonads, with ideal monad coproducts and cofree comonad products

- Higher-order functors, adjunctions, monads, and comonads that work over functors and map natural transformations

- Indexed (co)monads, including the well-known implementations for indexed state and delimited continuations

- Hyperfunctions

- Multiple functor composition operators to support (co)monad/(co)pointed functor composition and adjunction based (co)monads without ambiguity

Future plans

- Zippered comonadic automata

- A suite of (bi)functor type-level combinators inspired by similar term-level combinators such as On, Ap, and Join to help see the connections between different (co)monads and to simplify the taking of type-level derivatives.

- A "cofib" example suite to demonstrate programming with comonads.

- Better documentation

Further reading

http://comonad.com/haskell/category-extras/

5.10.4 IOSpec

Report by:	Wouter Swierstra
Status:	active development

The 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, analyze, or test impure code.

The IOSpec library is essentially a drop-in replacement for several other modules, such as Data.IORef and Control.Concurrent. Once you are satisfied that your functions are reasonably well-behaved with respect to the pure specification, you can drop the IOSpec import in favor of the "real" IO modules. The ideas underlying the previous version are described by a 2007 Haskell Workshop paper.

The latest version, however, supports several exciting new features. Besides providing a pure specification of STM, it allows you to build your own pure IO monad à la carte — allowing you to be explicit about which effects your program is using.

In the next major release, I would like to incorporate efficiency improvements suggested by Janis Voigtländer and allow the extraction of an IO computation from its pure counterpart.

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

Further reading

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

5.11 User interfaces

5.11.1 Gtk2Hs

Report by:	Peter Gavin
Participants:	Axel Simon, Duncan Coutts, many others
Status:	beta, actively developed

Gtk2Hs is a set of Haskell bindings to many of the libraries included in the Gtk+/Gnome platform. 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, FreeBSD, and Mac OS X using X11.

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
- cross-platform, multi-format multimedia playback with GStreamer
- novelties in darcs which will be in the next release:
  - completion of the model-view widgets with stores being implemented in Haskell
  - full drag-and-drop and clipboard support
- 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: GIO/GVfs, 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 (→ 2.9.1), Debian, and FreeBSD

The Gtk2HS library is continually being improved with new bindings, documentation, and bug fixes. Outside contributions are always welcome! We have recently released version 0.9.13, and are in the process of packaging a new version to be released soon.

In the future we hope to modularize Gtk2Hs and enable it to be built and distributed with Cabal and Hackage. This will enable people to just create, e.g., high-quality PDF documents using Cairo and Pango, performing image manipulation using Pixbuf and more. We also plan to bind more of the Gnome platform libraries, to allow compliant Gnome applications to be built entirely in Haskell. We also hope to implement a regular release schedule, have releases every six months or so. Since the last HCAR, Peter Gavin has joined the Gtk2Hs project as release manager which will allow us to do more regular releases and free resources for further completion of the binding.

Further reading

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

5.11.2 HQK

Report by:	Wolfgang Jeltsch
Participants:	Thomas Mönicke
Status:	early development

HQK is an effort to provide Haskell bindings to large parts of the Qt and KDE libraries. At the time of writing, we are developing a generator which shall produce most of the binding code automatically. In addition, we have developed a small Haskell module for accessing object-oriented libraries in a convenient way. This module also supports parts of Qt's signal-slot mechanism. In contrast to the original C++-based solution, type correctness of signal-slot connections is checked at compile time with our library.

We plan to develop a HQK GUI backend for the Functional Reactive Programming library Grapefruit (→ 6.5.1), thereby making Grapefruit multi-platform.

Further reading

http://haskell.org/haskellwiki/HQK

5.11.3 wxHaskell

Report by:	Jeremy O'Donoghue
Participants:	Shelarcy, Eric Kow, Mads Lindstroem, and others
Status:	beta, actively developed

The wxHaskell library provides Haskell bindings for a large subset of the wxWidgets library, which provides a cross-platform GUI library using native controls.

Using wxHaskell in a GUI project offers a number of benefits:

- Extensive and highly functional set of widgets, many of which have Haskell bindings which make development more declarative in feel.
- Native look and feel on all supported platforms (Windows, Mac OS X, and Linux), due to the use of platform native widgets in almost all cases.
- Simple deployment: only a small number of shared libraries need to be distributed with wxHaskell (e.g., one DLL on Windows).
- wxHaskell is used as the basis for a number of higher-level libraries including AutoForms http://autoforms.sourceforge.net/, wxGeneric, etc.

Over the past year, wxHaskell has seen considerable development, and is beginning the gain a small community outside of the core developers. We particularly appreciate the bugs found (and very often fixed) by members of the community.

The main changes in wxHaskell over the past year or so include:

- Release of version 0.10.3, including binary installers. This added support for recent versions of the underlying wxWidgets library and for recent versions of GHC. Prior to this release, getting started with wxHaskell was becoming difficult for new users.
- Support for many new widget types. We now support almost all of the widgets provided in the wxWidgets distribution, including user-contributed widgets.
- Many bugfixes.
- Addition of support for XML descriptions of GUI layouts using XRC. This has been the most requested feature by users for some time, and has just been committed to the Darcs repository. We anticipate a new release with binary installers in the next few weeks, to make this feature more widely available.
- Work on porting to the GHC 6.10 release — wxHaskell in Darcs repository compiles out of the box on 6.10 release candidates.
- Improvements to Cabal support, with the intention of providing Cabal installable packages for wxcore and wx.
- Main repository moved from darcs.haskell.org to code.haskell.org.
- Dropped support for versions of wxWidgets prior to 2.8. We currently support building against wxWidgets 2.8.x and 2.9.x. This has resulted in a (small) number of backward incompatible API changes, compared to earlier versions.

We are working on a tutorial to complement the Gtk2Hs chapter in the forthcoming "Real World Haskell" book, so that interested developers can work through developing a GUI for wxHaskell following a similar sequence.

Further reading

- News, downloads, documentation and tutorials: http://haskell.org/haskellwiki/WxHaskell
- Development version: darcs get http://code.haskell.org/wxhaskell/
- Binary packages: http://haskell.org/haskellwiki/WxHaskell/Download

5.11.4 Shellac

Report by:	Robert Dockins
Status:	beta, maintained

Shellac is a framework for building read-eval-print style shells. Shells are created by declaratively defining a set of shell commands and an evaluation function. Shellac supports multiple shell backends, including a "basic" backend, which uses only Haskell IO primitives, and a full featured "readline" backend based on the the Haskell readline bindings found in the standard libraries.

This library attempts to allow users to write shells in a declarative way and still enjoy the advanced features that may be available from a powerful line editing package like readline.

Shellac is available from Hackage, as are the related Shellac-readline, Shellac-editline, and Shellac-compatline packages. The readline and editline packages provide Shellac backends for readline and editline, respectively. The compatline package is a thin wrapper for either the readline or editline package, depending on availability at build-time.

Shellac has been successfully used by several independent projects and the API is now fairly stable.

Further reading

http://www.cs.princeton.edu/~rdockins/shellac/home

5.11.5 Haskeline

Report by:	Judah Jacobson
Status:	active development

The Haskeline library provides a user interface for line input in command-line programs. It is similar in purpose to readline or editline, but is written in Haskell and aims to be more easily integrated into other Haskell programs. A simple, monadic API allows this library to provide guarantees such as restoration of the terminal settings on exit and responsiveness to control-c events.

In its latest release (0.3.2), Haskeline supports Unicode and runs both on the native Windows console and on POSIX-compatible systems. Its rich line-editing interface is user-customizable and includes emacs and vi modes, history recall and incremental search, undo support, and custom tab completion functions.

Plans for further development include adding even more features to the user interface and continuing to expand cross-platform support.

Further reading

- Wiki and bug tracker: http://trac.haskell.org/haskeline
- Releases: http://hackage.haskell.org/cgi-bin/hackage-scripts/package/haskeline

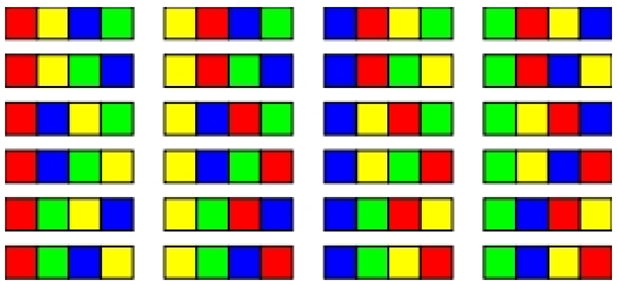
5.12 Graphics

5.12.1 diagrams

Report by:	Brent Yorgey
Participants:	Dougal Stanton
Status:	active development

The diagrams library provides an embedded domain-specific language for creating simple pictures and diagrams, built on top of the Cairo rendering engine. Values of type `Diagram` are built up in a compositional style from various primitives and combinators, and can be rendered to a physical medium, such as a file in PNG, PS, PDF, or SVG format.

For example, consider the following diagram to illustrate the 24 permutations of four objects:



The diagrams library was used to create this diagram with very little effort (about ten lines of Haskell, including the code to actually generate permutations). The source code for this diagram, as well as other examples and further resources, can be found at http://code.haskell.org/diagrams/.

Version 0.2, which adds support for paths, polygons, PDF, PS, and SVG output, text, more alignment modes, and additional drawing attributes, will be released soon! Features planned for future versions include grid and tree layouts, connectors, and animation support. New contributors and testers welcome!

Further reading

- http://code.haskell.org/diagrams/
- http://byorgey.wordpress.com/2008/04/30/new-haskell-diagrams-library/

5.12.2 FieldTrip

Report by:	Conal Elliott
Status:	active development

FieldTrip is a library for functional 3D graphics. It is intended for building static, animated, and interactive 3D geometry, efficient enough for real-time synthesis and display. Since FieldTrip is functional, one describes what models are, not how to render them (being rather than doing).

Surfaces are described as functions from 2D space to 3D space. As such, they are intrinsically curved rather than faceted. Surface rendering tessellates adaptively, caching tessellations in an efficient, infinite data structure (from the MemoTrie library (→ 5.7.7)) for reuse. Surface normals are computed automatically and exactly, using the derivative tools in the vector-space library (→ 5.6.4).

FieldTrip contains no support for animation, because it can be used with the Reactive library (→ 6.5.2) for functional reactive programming (and possibly other animation frameworks). By design, FieldTrip is completely orthogonal to any formulation or implementation of FRP.

Further reading

http://haskell.org/haskellwiki/FieldTrip

5.13 Music

5.13.1 YampaSynth

Report by:	George Giorgidze
Status:	Experimental

YampaSynth is a purely functional framework for programming modular synthesizers in Haskell using Yampa, a domain specific language embedded in Haskell for programming hybrid systems. A synthesizer, be it a hardware instrument or a pure software implementation, as here, is said to be modular if it provides sound-generating and sound-shaping components that can be interconnected in arbitrary ways.

Basic sound-generating and sound-shaping modules have been implemented, e.g., oscillator, amplifier, mixer, envelope generator, filter, etc. These modules are used to develop example applications:

- **yampasynth-wav** is an application which synthesizes MIDI music and writes the result into a WAVE audio file.

- **yampasynth-openal** is an application which synthesizes MIDI music and sends audio data in real-time to a sound card. We use a Haskell binding of the OpenAL library as an interface to audio hardware.

- **yampasynth-gtk** is an application with a simple graphical user interface that allows you to play music with various instruments in real-time using the keyboard of your computer. We use a Haskell binding of the GTK library for GUI programming, and a Haskell binding of the OpenAL library as an interface to audio hardware.

The source code, together with example applications, has been cabalized and is available under the BSD3 license.

Future plans

We would like to see a richer collection of sound-generating and sound-shaping modules in the framework, and complete implementation of MIDI, SoundFont, and related standards. However, one might find some other interesting continuation of the work. We are open for suggestions and would be happy if someone wishes to collaborate.

Further reading

- Related papers, slides, demos, and talks are available from my homepage
- YampaSynth Cabal package on Hackage
- HCodecs is a supporting library which provides functions to read, write, and manipulate MIDI, WAVE, and SoundFont2 multimedia files. It is written entirely in Haskell.

5.13.2 Haskore revision

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

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

1. The `Music` data type has been generalized in the style of Hudak's "polymorphic temporal media." It has been made abstract by providing functions that operate on it.
2. 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.
3. Creation of CSound orchestra files in a functional style including feedback and signal processors with multiple outputs.
4. Support for the software synthesizer SuperCollider both in real-time and non-real-time mode through the Haskell interface by Rohan Drape.
5. Conversion between MIDI file and Haskore representation of Music. Real-time MIDI is supported via ALSA on Linux.
6. A package for lists of events with time information has been factored out, as well as a package for non-negative numbers, which occur as time differences in event lists.
7. 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 (→ 6.6.1).)

Future plans

There is an ongoing effort by Paul Hudak to rewrite Haskore targeting at education.

Further reading

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

5.14 Web and XML programming

5.14.1 Haskell XML Toolbox

Report by:	Uwe Schmidt
Status:	seventh major release (current release: 8.1.1)

Description

The Haskell XML Toolbox (HXT) 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 one for Relax NG schemas.

The Haskell XML Toolbox is based on the ideas of HaXml (→ 5.14.2) and HXML, but introduces a more general approach for processing XML with Haskell. The processing model is based on arrows. The arrow interface is more flexible than the filter approach taken in the earlier HXT versions and in HaXml. It is also safer; type checking of combinators becomes possible with the arrow approach.

HXT consists of two packages, the old first approach (hxt-filter) based on filters and the newer and more flexible and save approach using arrows (hxt). The old package hxt-filter, will further be maintained to work with the latest ghc version, but new development will only be done with the arrow based hxt package.

Features

- Validating XML parser
- Very liberal HTML parser
- Lightweight lazy parser for XML/HTML based on Tagsoup (→ 5.14.3)
- Easy de-/serialization between native Haskell data and XML by pickler and pickler combinators
- XPath support

- Full Unicode support
- Support for XML namespaces
- Cabal package support for GHC
- HTTP access via Haskell bindings to libcurl
- Tested with W3C XML validation suite
- Example programs
- Relax NG schema validator
- An HXT Cookbook for using the toolbox and the arrow interface
- Basic XSLT support
- Darcs repository with current development version (8.1.2) under http://darcs2.fh-wedel.de/hxt

Current Work

Currently mainly maintenance work is done. This includes conversion to work with ghc.6.10 and space and runtime optimizations.

It is planned to further develop and extend the validation part with Relax NG and the conversion from/to Haskell internal data. The pickler approach used in that task can be extended to derive DTDs, Relax NG Schemas or XML Schemas for Validation of the external XML representation.

The HXT library will be extensively used in the Holumbus project (→ 6.3.1).

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 theses 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 available in the Haskell Wiki (http://www.haskell.org/haskellwiki/HXT).

5.14.2 HaXml

Report by:	Malcolm Wallace
Status:	stable, maintained

HaXml provides many facilities for using XML from Haskell. The public stable release on Hackage is 1.13.3, with support for building via Cabal for ghc-6.8.x.

The development version (currently at 1.19.4, also available on Hackage or through a Darcs repository) includes a much-requested lazy parser and a SAX-like streaming parser. Some minor work still(!) remains to tidy things up before the development version is tagged and released as stable.

The lazy parser combinators used by HaXml now live in 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

5.14.3 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 (→ 5.1) listings, and is used in the next version of Hoogle (→ 4.4.1).

Work continues on the API of tagsoup, and the implementation. Lots of people have made use of tagsoup in their applications, generating lots of valuable feedback.

Further reading

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

5.15 System

5.15.1 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.

As file and directory notification is available for many operating systems, upcoming work will include to try to find a common API that could be shared for all platforms. Most recent work has been to see what is possible to do under Microsoft Windows, and finding a suitable API for both platforms. This has been a joint work with Niklas Broberg. We are still looking for contributors to *BSD and Mac OS X. If you are interested, contact us.

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/

5.15.2 hlibev

Report by:	Aycan Irican
Participants:	Evrim Ulu
Status:	Unstable

hlibev is an FFI wrapper for "libev event loop". Currently we only implemented IO and Timer event types on the Debian GNU/Linux platform. We implemented a simple http responder to see it's performance. You can get it from hackage.

Further reading

http://software.schmorp.de/pkg/libev.html

6 Applications and Projects

6.1 For the Masses

6.1.1 Darcs

Report by:	Eric Kow
Participants:	Jason Dagit, Simon Michael
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 a very easy to use tool for every day use because it follows the principle of keeping simple things simple.

We have recently released darcs 2.1, providing better HTTP support and fixes to our use of the pending patch. The new version also creates darcs 2 format repositories by default, as darcs 2 is starting to be more widely used and shipped by major package distributors. This change will bring the new improved conflicts handling and merging semantics to a wider audience.

In the bigger picture, the past six months have been particularly invigorating for the darcs team. GHC is moving to git from darcs and we are applying their feedback to make darcs better. The decision to switch has been painful, but it has spurred us into more vigorous action in building up the darcs team. As a result of this switch, we have undertaken several long term projects to improve our development practices and grow the darcs community:

1. Increased automation. We now use automated buildbots for major supported platforms. The buildbots run our ever-growing suite of test cases, including a new performance regression suite.

2. Code documentation. We have begun an effort to provide better developer documentation for the darcs. We are integrating many fine Haskell tools, such as Haddock and Hoogle at http://darcs.net/api-doc.

3. Darcs Weekly News. Inspired from Haskell Weekly News, we are using this bulletin to help on-lookers to catch up with recent discussions, issues resolved, and patches applied. See http://wiki.darcs.net/DarcsWiki/DarcsWeeklyNews.

4. Regular hacking sprints. The first darcs hacking sprint was held on 25–26 October and was a great success! We hope to host similar sprints twice a year as a means for experienced developers to concentrate on darcs, and also for newcomers to get to grips with the code.

These practices should help us continue to steadily improve darcs, and we are always open to contributions. Haskell hackers, we need your help!

Darcs is free software licensed under the GNU GPL.

Further reading

http://darcs.net

6.1.2 xmonad

Report by:	Don Stewart
Status:	active development

xmonad is a tiling window manager for X. Windows are arranged automatically to tile the screen without gaps or overlap, maximizing screen use. Window manager features are accessible from the keyboard: a mouse is optional. xmonad is written, configured, and extensible in Haskell. Custom layout algorithms, key bindings, and other extensions may be written by the user in config files. Layouts are applied dynamically, and different layouts may be used on each workspace. Xinerama is fully supported, allowing windows to be tiled on several physical screens.

The new release 0.7 of xmonad added full support for the GNOME and KDE desktops, and adoption continues to grow, with binary packages of xmonad available for all major distributions.

Further reading

- Homepage: http://xmonad.org/
- Darcs source:
  `darcs get` http://code.haskell.org/xmonad
- IRC channel: `#xmonad @ irc.freenode.org`
- Mailing list: ⟨xmonad@haskell.org⟩

6.2 Education

6.2.1 Exercise Assistants

Report by:	Bastiaan Heeren
Participants:	Alex Gerdes, Johan Jeuring, Josje Lodder, Harrie Passier, Sylvia Stuurman
Status:	experimental, active development

At the Open Universiteit Nederland we are building a collection of tools that support students in solving exercises incrementally by checking intermediate steps. These tools are written in Haskell. The distinguishing feature of our tools is the detailed feedback that they provide, on several levels. For example, we have an online exercise assistant that helps to rewrite logical expressions into disjunctive normal form. Students get instant feedback when solving an exercise, and can ask for a hint at any point in the derivation. Other areas covered by our tools are solving linear equations, reducing matrices to echelon normal form, and simplifying expressions in relation algebra. We have just started to explore exercise assistants for learning how to program in a (functional) programming language.

For each exercise domain, we have formulated a set of rewrite rules, as well as a number of unsound (or buggy) rules to catch common mistakes. With these rules we can check all intermediate steps submitted by the user. We use datatype-generic rewriting technology, which makes it possible to observe the rules (e.g., for generating documentation, or for automated testing). We also defined strategies for solving the exercises. A strategy dictates in which order the rules have to be applied to reach the solution, and such a strategy takes the form of a context-free grammar. These strategies support us in reporting helpful and informative feedback.

We are offering our tools and our strategies as web-services to other e-learning tools and environments, such as MathDox and LeActiveMath. We have collected data on student interactions with our system, and we are currently analyzing this to further improve our exercise assistants. For the near future, we have scheduled more sessions with students from our university to validate our approach. We also plan to make more use of generic programming techniques to support exercises from many more, different domains.

An online prototype version for rewriting logical expressions is available and can be accessed from our project page.

Further reading

- http://ideas.cs.uu.nl/trac
- Strategies for exercises. Bastiaan Heeren, Johan Jeuring, Arthur van Leeuwen, and Alex Gerdes. International Conference on Mathematical Knowledge Management (MKM'08).
- Recognizing Strategies. Bastiaan Heeren, Johan Jeuring. Reduction Strategies in Rewriting and Programming (WRS'08)
- A Lightweight Approach to Datatype-Generic Rewriting. Thomas van Noort, Alexey Rodriguez, Stefan Holdermans, Johan Jeuring, and Bastiaan Heeren. Workshop on Generic Programming (WGP'08).

6.2.2 Holmes, plagiarism detection for Haskell

Report by:	Jurriaan Hage
Participants:	Brian Vermeer

Years ago, Jurriaan Hage developed Marble to detect plagiarism among Java programs. Marble was written in Perl, takes just 660 lines of code and comments, and does the job well. The techniques used there, however, do not work well for Haskell, which is why a master thesis project was started, starring Brian Vermeer as the master student, to see if we can come up with a working system to discover plagiarism among Haskell programs. We are fortunate to have a large group of students each year that try their hand at our functional programming course (120-130 per year), and we have all the loggings of Helium that we hope can help us tell whether the system finds enough plagiarism cases. The basic idea is to implement as many metrics as possible, and to see, empirically, which combination of metrics scores well enough for our purposes. The implementation will be made in Haskell. One of the things that we are particularly keen about, is to make sure that for assignments in which students are given a large part of the solution and they only need to fill in the missing parts, we still obtain good results.

In May this was the plan, and it still is.

6.2.3 Geordi IRC C++ eval bot

Report by:	Eelis van der Weegen
Status:	mature

Geordi is an IRC bot that compiles and (optionally) runs C++ code snippets. It has proved to be a very useful tool when teaching and discussing C++ on IRC. It is written in Haskell, and, being deployed on C++ channels at most of the big IRC networks, has the sneaky side-effect of getting some C++ers interested in Haskell ;-).

Snapshots and Darcs repository can be found at the homepage.

Further reading

http://www.eelis.net/geordi/

6.2.4 Lambda Shell

Report by:	Robert Dockins
Status:	beta, maintained

The Lambda Shell is a feature-rich shell environment and command-line tool for evaluating terms of the pure, untyped lambda calculus. The Lambda Shell builds on the shell creation framework Shellac (→ 5.11.4), and showcases most of Shellac's features.

Features of the Lamba Shell include:

- Evaluate lambda terms directly from the shell prompt using normal or applicative order. In normal order, one can evaluate to normal form, head normal form, or weak head normal form.
- Define aliases for lambda terms using a top level, non-recursive "let" construct.
- Show traces of term evaluation, or dump the trace to a file.
- Count the number of reductions when evaluating terms.
- Test two lambda terms for beta-equivalence (that is; if two terms, when evaluated to normal form, are alpha equivalent).
- Programs can be entered from the command line (using the `-e` option) or piped into stdin (using the `-s` option).
- Perform continuation passing style (CPS) transforms on terms before evaluation using the double-bracket syntax, e.g., "[[five]]".

The Lambda Shell was written as a showcase and textbook example for how to use the Shellac shell-creation library. However, it can also be used to gain a better understanding of the pure lambda calculus.

Further reading

- http://http://www.cs.princeton.edu/~rdockins/lambda/home
- http://http://www.cs.princeton.edu/~rdockins/shellac/home

6.2.5 INblobs — Interaction Nets interpreter

Report by:	Miguel Vilaca
Participants:	Daniel Mendes
Status:	active, maintained
Portability:	portable (depends on wxHaskell)

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

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

New features

- automatic transformation of textual functional terms into interaction nets
- generation of textual descriptions allowing the use of INblobs as an editor/frontend for textual IN compilers
- *Valid IN System* check
- minor changes for better usability

Further reading

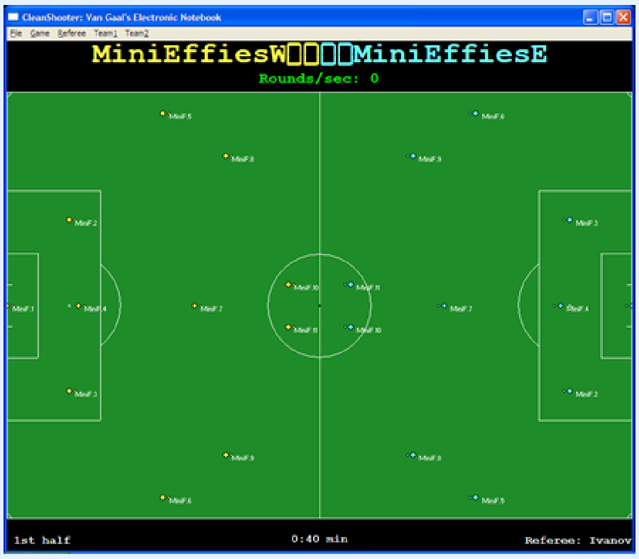
- Homepage:
  http://haskell.di.uminho.pt/jmvilaca/INblobs/
- also available in Hackage (http://hackage.haskell.org/cgi-bin/hackage-scripts/package/INblobs)
- Blobs: http://www.cs.york.ac.uk/fp/darcs/Blobs

6.2.6 Soccer-Fun

Report by:	Peter Achten
Status:	active development

Soccer-Fun is an educational project to stimulate functional programming by thinking about, designing, implementing, running, and competing with the brains of football players! It is open for participation by everybody who likes to contribute. It is not restricted to a particular functional programming language. The current implementation is in Clean (→3.2.3).

With Soccer-Fun you can program footballer brains and run them in the environment that you see here:


CleanShooter: Van Gaal's Electronic Notebook
File Game Referee Team1 Team2
MiniEffiesW□□□□MiniEffiesE
Rounds/sec: 0
1st half
0:40 min
Referee: Ivanov

The brain of a footballer is really a function, as was once explained by Johan Cruijff himself:

> "If I play the ball and want to pass it to someone, then I need to consider my guardian, the wind, the grass, and the velocity with which players are moving. We compute the force with which to kick and its direction within a tenth of a second. It takes the computer two minutes to do the same!" (De Tijd, 2 mei 1987)

The brain that you program has a different type than the one above. It takes five parameters: the referee actions, the football (if freely available), all players on the field except you, you, and your current memory. Using these parameters, we compute a footballer's action such as moving, kicking the ball, as well as a new memory value.

In a nutshell, it is your task to create a team of footballers, equip them with the brains that you have created, and see whether you can beat other teams!

Future plans

There are many plans for the future:

- Use TCP/IP to allow individual footballers to "hook in" the framework. Then, footballers can be programmed in arbitrary programming languages and join Soccer-Fun.
- Related: in the current framework, the code of all footballers is included in the framework. What about using interpreter technology, like Jan Martin Jansen's SAPL?
- Also related: the semantics is less suited for an "individual" footballer approach, because individual actions can affect other players (think of gaining the ball). A more fine grained semantic model can be developed to allow individual actions to be performed.
- Currently, all footballers and the referee are panoptic, which is not very realistic. Programming brains becomes much more challenging if we limit the viewing range of footballers. In that situation, footballers need to maintain some sort of mental model of the whereabouts of all players.
- The current rendering is plain 2D. It would be more informative to use a simple 2.5D rendering.

Further reading

- http://www.cs.ru.nl/P.Achten/SoccerFun/SoccerFun.html
- http://www.st.cs.ru.nl/papers/2008/achp08-FDPE08-SoccerFun.pdf

6.3 Web Development

6.3.1 Holumbus Search Engine Framework

Report by:	Uwe Schmidt
Participants:	Timo B. Hübel, Sebastian Reese, Sebastian Schlatt, Stefan Schmidt
Status:	first release

Description

The Holumbus framework consists of a set of modules and tools for creating fast, flexible, and highly customizable search engines with Haskell. The framework consists of two main parts. The first part is the indexer for extracting the data of a given type of documents, e.g., documents of a web site, and store it in an appropriate index. The second part is the search engine for querying the index.

An instance of the Holumbus framework is the Haskell API search engine Hayoo! (http://holumbus.fh-wedel.de/hayoo/). The web interface for Hayoo! is implemented with the Janus web server, written in Haskell and based on HXT (→ 5.14.1).

Features

- Highly configurable crawler module for flexible indexing of structured data
- Customizable index structure for an effective search
- *find as you type* search
- Suggestions
- Fuzzy queries
- Customizable result ranking
- Index structure designed for distributed search
- Darcs repository with current development version under http://darcs2.fh-wedel.de/holumbus

Current Work

Currently the indexer and search module will be used and extended to support the Hayoo! engine for searching the hackage package library (http://holumbus.fh-wedel.de/hayoo/hayoo.html).

Stefan Schmidt will finish his master thesis developing a framework for distributed computing using the Google *map–reduce* approach at the end of this year. This extension will be used in the future to recompute and update the Hayoo! and other indexes much faster on a whole cluster of machines.

The design of this *map–reduce* framework and library is independent of the Holumbus and Hayoo! applications. It will be possible with this system to distribute arbitrary task on a cluster of machines. It will only be necessary to separate an application into independent parts and then support a simple interface to the map-reduce framework by an instantiation of a Haskell class.

Further reading

The Holumbus web page (http://holumbus.fh-wedel.de/) includes downloads, Darcs web interface, current status, requirements, and documentation. Timo Hübel's Master Thesis describing the Holumbus index structure and the search engine is available at http://holumbus.fh-wedel.de/branches/develop/doc/thesis-searching.pdf.

6.3.2 Top Writer

Report by:	Jon Strait
Status:	experimental, active development

Top Writer is a web application for technical writers to easily edit and assemble topic-oriented user guides and other high level reference works. Application users edit within a structured framework, using meaningful application elements to chunk and contain content. Users can extend the application with their own elements and rules, if needed. Delivery of content is meant to be multi-format, with each format having separate templating rules.

The server part of the application is coded in Haskell using FastCGI on a lighttpd server and using the HDBC library connecting to a PostgreSQL database server. The client web browser part heavily uses the jQuery Javascript toolkit.

Future plans

Currently, the focus for delivering output is on generated HTML, but plans are also to generate PDF and any other format that is reasonable. Other work is focused on collaborative features, allowing multiple clients to share projects, edit contained documents concurrently, and see other project member's changes immediately. The more element-like structure for editing a document can facilitate this, removing the complexity of having to consider overlapping changes.

Further reading

http://www.moonloop.net/topwriter

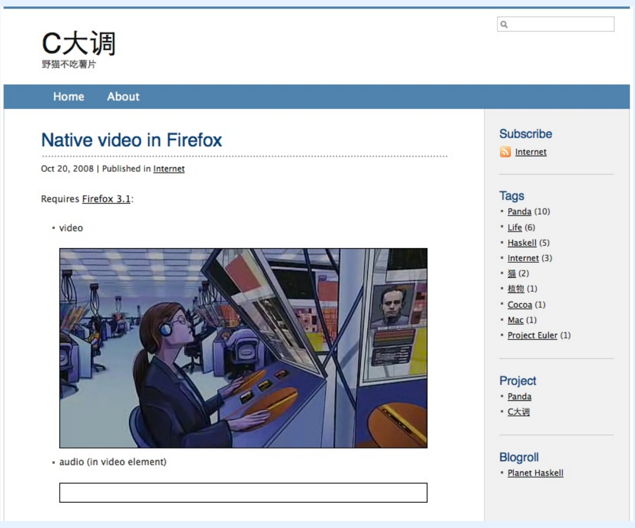
6.3.3 Panda blog engine

Report by:	Jinjing Wang
Status:	experimental

Panda is a simple static blog engine written in Haskell. It uses plain text as data source and (preferably) an SCM for data management.

Contrary to main stream functional programs, Panda embraces object oriented designs and idioms, in particular, it strictly follows the MVC pattern and programmed in way that looks like traditional object oriented languages. Despite this coding style, underneath it is still Haskell, so it benefits from static typing, purity and laziness.

Panda is free software under GPL, future development will focus on usability (blindly follows the KISS principle) and hackability (empower the user to extend/customize the framework).



Native video in Firefox

Oct 20, 2008 | Published in Internet

Requires Firefox 3.1:

- video
- audio (in video element)

Further reading

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

6.3.4 InputYourData.com

Report by:	Enzo Haussecker
Status:	beta

I would like to announce the publication of InputYourData.com (beta) — the online resource tool for financial, mathematical, and scientific calculations. All web applications found at http://inputyourdata.com/ are written solely in Haskell and based on the Network.CGI framework.

This website began as an experiment to familiarize myself with the monadic features of Haskell and their use in web programming. Namely, the mapping and manipulation of user inputs as typed objects. Through these experiments I found that Haskell allows for an efficient system where a variety of operations can be performed while minimizing as many resources (such as time and memory space) as possible.

I am now interested in developing a similar type of website, except wiki style — where all web applications are created by the user. Essentially, I am designing a web application where users can symbolically declare the arguments of a function and that function's call based on arbitrary variables. For example, say a user would like to create a web application to compute the roots of a second degree polynomial. He/she would simply declare the arguments to her function — three complex numbers a b and c, and the call of that function — $(-b \pm \text{sqrt } (b^2 - 4 * a * c))/(2 * a)$. The product of that user's inputs will render a web application that looks similar to http://inputyourdata.com/cgi-bin/quadratic.cgi (all text is to be updated by the user as well). As one could imagine, other, more complex functions involving vectors, matrices, stock prices, and other arguments can also be defined in terms of arbitrary variables and declared as inputs to my wiki-style web application.

If you are intrigued by this project and you have substantial experience in designing Haskell-based web applications, please send me (⟨ehaussecker@gmail.com⟩) your resume and a brief summery of why you are interested.

6.3.5 Hircules

Report by:	Jens Petersen
Status:	hibernating

Hircules is an IRC client built on top of gtk2hs (→ 5.11.1).

Not too much has happened recently except an update in cvs to make it build with ghc-6.8 (i.e., change from Data.FiniteMap to Data.Map).

I am planning finally to move the code within this year to code.haskell.org, move to cabal and make a new working release in Hackage; maybe making use of the Haskell irc library. I still would like to add support for multiple irc connections and get other people involved.

Further reading

http://www.haskell.org/hircules/

6.4 Data Management and Visualization

6.4.1 Pandoc

Report by:	John MacFarlane
Participants:	Recai Oktaş, Andrea Rossato, Peter Wang
Status:	active development

Pandoc aspires to be the swiss army knife of text markup formats: it can read markdown and (with some limitations) HTML, LaTeX, and reStructuredText, and it can write markdown, reStructuredText, HTML, DocBook XML, OpenDocument XML, ODT, RTF, groff man, MediaWiki markup, GNU Texinfo, LaTeX, ConTeXt, and S5. Pandoc's markdown syntax includes extensions for LaTeX math, tables, definition lists, footnotes, and more.

The latest release, 1.0.0.1, features

- New writers for GNU Texinfo (thanks to Peter Wang), OpenDocument XML (thanks to Andrea Rossato), ODT (OpenOffice document), and MediaWiki.
- A new delimited code block syntax, with optional syntax highlighting (using the highlighting-kate library).
- Handy generic functions for querying and transforming documents without lots of boilerplate (thanks to Andrea Rossato)
- A cleaner build system: Pandoc can now be built as a regular Cabal package, and can be installed using cabal-install.
- Better support for math, including display math
- HTML sanitizing for use in web applications
- Optional (and experimental) integration with Andrea Rossato's hs-citeproc library, for automatic generation of citations and bibliography in any of pandoc's output formats.

Future plans include improved citation support and better support for writing literate Haskell in markdown. Contributions are welcome.

Further reading

http://johnmacfarlane.net/pandoc/

6.4.2 tiddlyisar

Report by:	Slawomir Kolodynski
Status:	under development

tiddlyisar is a tool for generating TiddlyWiki renderings of IsarMathLib source. IsarMathLib is a library of mathematical proofs formally verified by the Isabelle/ZF theorem proving environment. The tiddlyisar tool parses IsarMathLib source and generates TiddlyWiki markup text. The generated view features jsMath based mathematical symbols, cross referenced theorems, and structured proofs expanded on request. The rendering can be viewed on the Tiddly Formal Math site. tiddlyisar is included in the IsarMathLib distribution under GPLv3 license. The source can be browsed at the IsarMathLib Subversion repository URL provided below.

Further reading

- http://savannah.nongnu.org/projects/isarmathlib
- http://www.cl.cam.ac.uk/research/hvg/Isabelle/
- http://formalmath.tiddlyspot.com
- http://svn.savannah.gnu.org/viewvc/trunk/isarmathlib/tiddlyisar/?root=isarmathlib

6.4.3 Emping

Report by:	Hans van Thiel

Emping 0.5 has been released. Emping is a (prototype of) a tool for the analysis of multi-variate nominal data. For example, in a table of 8000 mushrooms and 20 attributes, constructed from a field guide, the tool finds which attribute-values determine whether a mushroom is edible or poisonous. But Emping finds

not only single factors, but also pairs, triples, and all other combinations which distinguish between the consequent values. Such reduced rules are generalizations of rows in the original table, so r1 could stand for originals a,b,c and r2 for a,b. In that case r2 implies r1 or, conversely, r1 entails r2. The reductions are partially ordered. Emping also finds all such dependencies, including the equivalences where different reductions stand for the same original rules. New in Emping 0.5 is that, thanks to the functional graph library which comes with GHC, these dependencies are now expressed in a Graphviz format and can be shown with a Graphviz reader. Also new is the sort by length of the reduced rules, and of the reduced rules in each equivalence class. This makes the results much more readable. Starting in 0.4, it is now also possible to have blank fields, but this feature has only been summarily tested. The Gtk2Hs based GUI (→5.11.1), first introduced in version 0.4, has been improved in Emping 0.5. Data tables, as well as output tables of reduced rules and graph legends, all use the default CSV format of the Open Office Calc spreadsheet.

Further reading

See http://home.telfort.nl/sp969709/emp/empug.html for more, including two white papers and downloads.

6.4.4 HaExcel — From Spreadsheets to Relational Databases and Back

Report by:	Jácome Cunha
Participants:	João Saraiva, Joost Visser
Status:	Unstable, work in progress

HaExcel is a framework to manipulate, transform, and query spreadsheets. It is composed by a generic/reusable library to map spreadsheets into relational database models and back: this library contains an algebraic data type to model a (generic) spreadsheet and functions to transform it into a relational model and vice versa. Such functions implement the refinement rules introduced in paper "From Spreadsheets to Relational Databases and Back". The library includes two code generator functions: one that produces the SQL code to create and populate the database, and a function that generates Excel/Gnumeric code to map the database back into a spreadsheet. A MySQL database can also be created and manipulated using this library under HaskellDB.

The tool also contains a front-end to read spreadsheets in the Excel and Gnumeric formats: the front-end reads spreadsheets in portable XML documents using the *UMinho Haskell Libraries*. We reuse the spatial logic algorithms from the UCheck project to discover the tables stored in the spreadsheet.

Finally, two spreadsheet tools are available: a batch and an online tool that allows the users to read, transform, refactor, and query spreadsheets.

The sources and the online tool are available from the project home page.

We are currently exploring foreign key constraints from their detection to their migration to the generated spreadsheet. Another topic under study is the direct integration of the framework in Excel implemented as an Excel plug-in.

Further reading

http://haskell.di.uminho.pt/jacome/index.html

6.4.5 Between Types and Tables

Report by:	Bas Lijnse
Participants:	Rinus Plasmeijer
Status:	experimental

My master's thesis project aimed at bridging the gap between data stored in relational databases and data structures in a functional language. We have developed a method to derive both a relational database schema and a set of data types in Clean (→3.2.3) from an ORM (Object Role Modeling) model. We then realized an automatic mapping between values of those Clean types and their counterparts in the relational database using Clean's generic programming mechanism. We defined a generic library which provides the basic CRUD (Create, Read, Update, Delete) operations for any conceptual entity defined in an ORM model.

Future plans

Currently, the library is a proof of concept that only works with Clean on Linux and a MySQL database. However, we intend to integrate this library with the work on dynamic workflow specifications in the iTask system (→6.8.4) somewhere in the (near) future.

Further reading

http://www.st.cs.ru.nl/papers/2008/lijb08-BetweenTypesAndTablesMasterThesis.pdf

6.4.6 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). The category of supported metrics are size, complexity, structural and disambiguation. The disambiguation metrics are applicable to the SDF formalism only. Metrics output is a textual report or in Comma Separated Value format. The additional analyses implemented are graph visualization of the immediate successor graph, transitive closure graph and strongly-connected components graph outputted in DOT format, and visualization of the non-singleton levels of a grammar.

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*.

The tool was used successfully in a grammar engineering work presented in the paper *A Case Study in Grammar Engineering*.

Future plans

Efforts are underway to develop functionalities to compute quality profiles based on histograms. Furthermore, more metrics will be added, and a web-interface is planned.

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://sdfmetz.googlecode.com) includes the tool source code, and pointers to relevant work about grammar metrication.

6.5 Functional Reactive Programming

6.5.1 Grapefruit

Report by:	Wolfgang Jeltsch
Participants:	Matthias Reisner
Status:	provisional

Grapefruit is a Functional Reactive Programming library with special support for graphical user interfaces and animated graphics.

Reactive and interactive systems are described as networks of basic components and built using arrow combinators. The parts of such networks communicate via signals. There exist three kinds of them: discrete signals, continuous signals, and segmented signals. Discrete signals describe event sequences and continuous signals denote values that vary over time. Segmented signals also describe time-varying values. However, a segmented signal splits time into slices whereby a value change can only occur at the start of a new slice. Signals can be composed in a purely functional way.

At the moment of writing, we are changing Grapefruit's interface and implementation fundamentally. Our new implementation will support memoization of signals through ordinary **let**-bindings. The new interface will introduce era parameters for all signal types. An era parameter denotes (at compile time) the time slice during which the signal is active. Using era parameters, we can prevent signals from being started at different times. This is useful since otherwise a signal's behavior could depend on the starting time, leading to different behaviors of the same signal.

Grapefruit programs always cover the complete lifetimes of the systems in question. The developer is freed from dealing with technical details like object creation and event handler registration. This is in line with the declarative nature of Haskell, because it stresses the behavior of systems instead of how this behavior is achieved. On the other hand, we try hard to offer efficient execution of systems implemented with Grapefruit.

Grapefruit is currently based on Gtk2Hs (→ 5.11.1) and HOpenGL, but implementations on top of other GUI and graphics libraries are possible. The aim is to provide alternative implementations based on other GUI toolkits, so that a single application is able to integrate into multiple desktop environments. Concretely, we plan to offer an HQK (→ 5.11.2) GUI backend for Grapefruit.

Further reading

http://haskell.org/haskellwiki/Grapefruit

6.5.2 Reactive

Report by:	Conal Elliott
Status:	active development

Reactive is a simple foundation for functional reactive programming (FRP), including continuous, time-varying behaviors and compositional functional events. Some unusual features, relative to earlier FRP formulations and implementations:

- Much of the original interface is replaced by instances of standard type classes. In most cases, the denotational semantics of these instances is simple and inevitable, following from the principle of type class morphisms.
- The original idea of reactive behaviors is composed out of two simple notions:
  - *Reactive values* are temporally discrete and reactive. They have a purely data representation, and hence cache for free.
  - *Time functions* are temporally continuous and non-reactive.
- Reactive provides and builds on *functional futures*, which are time/value pairs with several handy type class instances. Futures allow one to conveniently compute with values before they can be known, with

a simple, purely functional semantics (no IO). Futures are polymorphic over both values *and* time, requiring only that time is ordered.

- A particularly useful type of time, based on Warren Burton's "improving values", reveals partial information in the form of lower bounds and minima, before the times can be known precisely. (Semantically non-flat.)
- Improving values are implemented on top of a semantically simple "unambiguous choice" operator, (see unamb (→ 5.3.7)).
- Reactive manages (I hope) to get the efficiency of data-driven computation with a (sort-of) demand-driven architecture. For that reason, Reactive is garbage-collector-friendly.

For the past few months, this work has been graciously supported by Anygma.

Further reading

http://haskell.org/haskellwiki/Reactive

6.5.3 Functional Hybrid Modeling

Report by:	George Giorgidze
Status:	Experimental

Under Henrik Nilsson's supervision I am working on a Functional Hybrid Modeling (FHM) project. The goal of the project is to design and implement a new language for non-causal, hybrid modeling and simulation of physical systems.

Causal modeling languages are closely related to synchronous data-flow languages. They model system behavior using ordinary differential equations (ODEs) in explicit form. That is, cause-effect relationship between variables (which are computed from which) must be explicitly specified by the modeler. In contrast, non-causal languages model system behavior using differential algebraic equations (DAEs) in implicit form, without specifying their causality. Inferring causality from usage context for simulation purposes is left to the compiler. The fact that the causality can be left implicit makes modeling in a non-causal language more declarative (the focus is on expressing the equations in a natural way, not on how to express them to enable simulation) and also makes the models much more reusable.

FHM is an approach to modeling which combines functional programming and non-causal modeling with the aim to improve state of the art of non-causal modeling languages. In particular, the FHM approach proposes modeling with first class model fragments (defined by continuous DAEs) using combinators for their composition and discrete switching. The key concepts of FHM originate from work on Functional Reactive Programming (FRP).

We are implementing Hydra, an FHM language, as a domain-specific language embedded in Haskell. The method of embedding employs quasiquoting, a new feature introduced in GHC 6.10, and enables modelers to use the domain specific syntax in their models. The present prototype implementation of Hydra enables modeling with first class model fragments, but only supports continuous systems. Implementing discrete switching combinators to enable hybrid modeling will be the next major step. The cabalized source code of the prototype is publicly available on-line under the open source BSD license.

Further reading

The source code and related papers, are available from my homepage.

6.6 Audio and Graphics

6.6.1 Audio signal processing

Report by:	Henning Thielemann
Status:	experimental, active development

In this project, audio signals are processed using pure Haskell code and the Numeric Prelude framework (→ 5.6.3). The highlights are:

- a basic signal synthesis backend for Haskore (→ 5.13.2),
- experimental structures for filter networks,
- basic audio signal processing, including some hard-coded frequency filters,
- support for 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,
- unlike other software synthesizer packages, there are not two global rates, namely control and audio rate. Instead there can be many different rates, and a signal processor can have different control and audio rate, in which case the internal processor control parameters are interpolated.
- frameworks 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. We tried hard to preserve the functional style of programming and do not need Arrows and according notation.

- We checked several low-level implementations in order to achieve reasonable speed. The standard list data structure is very convenient for programming, and especially the element-wise laziness allows for a range of very elegant implementations, but it is much too slow for signal processing. We complement this structure with a lazy `StorableVector` structure and a `StateT s Maybe a` generator.

- support for causal processes. Causal signal processes only depend on current and past data and thus are suitable for real-time processing (in contrast to a function like time reversal). These processes are modeled as `mapAccumL` like functions. Many important operations like function composition maintain the causality property. They are important in feedback loops where they statically warrant that no future data is accessed.

Further reading

- http://www.haskell.org/haskellwiki/Synthesizer
- http://dafx04.na.infn.it/WebProc/Proc/P_201.pdf

6.6.2 hsProcMusic

Report by:	Stephen Lavelle
Status:	Work ongoing

A collection of several different music-related Haskell programs, designed chiefly as compositional tools, rather than as music-generation tools. Most of the programs are, unfortunately, not very well documented. However, I would certainly welcome and respond to any requests to explain any of the code.

- Generate a 2-part melodic canon from a motivic seed. It implements a toy-version of first species counterpoint to generate canonical material.

- Analyze sets of melodies and produce candidates submelodies (including, optionally, slightly altered submelodies) for prospective polyphonic combination. This is related to a currently unrealized program to investigate cohomological aspects of counterpoint.

- Consonance-preserving map generator. Given two collections of pitch-classes, this program (with some harmonic assumptions) can generate all transformations from one to another that preserve relative consonance.

- A toy chord-progression generator based around Lerdahl's *Generative Theory of Tonal Music*. This is currently in active development, and I'm aiming to bulk out its harmonic capabilities over the following months.

The source code can be downloaded from my website.

Further reading

http://www.maths.tcd.ie/~icecube/tag/hsprocmusic/

6.6.3 Glome

Report by:	Jim Snow
Status:	experimental

Glome is a rendering engine for 3-D graphics, based on ray tracing. It was originally written in OCaml, but has since been ported (except for a few features) to Haskell, and most future development is likely to happen in Haskell.

It supports shadows and reflections, and base primitives include triangles, disks, boxes, cylinders, cones, spheres, boxes, and planes. More complex primitives can be made by grouping primitives, taking the Boolean difference or intersection of primitives, or by making transformed instances.

Input and output capabilities are limited. Input is accepted as NFF-format files, or scenes may also be hard-coded in Haskell. Output is via an OpenGL window.

Rendering speed is reasonably fast, but a little too slow for interactive graphics. Glome uses a Bounding Interval Hierarchy internally to reduce the number of ray-intersection tests, so that, in general, rendering time increases logarithmically with scene complexity rather than linearly.

Further reading

http://syn.cs.pdx.edu/~jsnow/glome/

6.6.4 easyVision

Report by:	Alberto Ruiz
Status:	experimental, active development

The *easyVision* project is a collection of libraries for computer vision and image processing. The goal is to write simple applications in an elegant, functional style, supported by optimized libraries for the low level expensive computations (Intel's IPP, HOpenGL, hmatrix (→ 5.3.5), MPlayer, etc.).

Recent developments include an improved parser for automatic IPP wrapper generation, some preliminary work on scale-invariant interest point detectors, detailed installation instructions, and the first sections of the tutorial.

Further reading

http://www.easyVision.googlepages.com

6.6.5 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.

Between versions 2.0 and 2.1 the software is largely the same on the outside but has undergone extensive changes inside. Most of this involved redesign with monad transformers.

photoname is on Hackage and can be acquired using darcs or other methods. See the project page below for more.

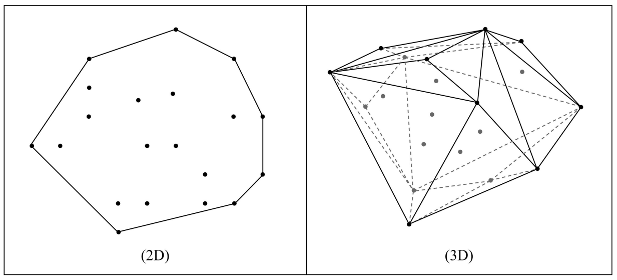
Further reading

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

6.6.6 Simplex-Based Spatial Operations

Report by:	Farid Karimipour
Participants:	Andrew U. Frank
Status:	active development

The project is to implement spatial operations independent of dimension. There is much need in computational geometry and related fields to extend 2D spatial operations to 3D and higher dimensions. Approaches designed for a specific dimension lead to different implementations for different dimensions. Following such approaches, the code for a package that supports spatial operations for both 2D and 3D cases is nearly two times the code size for 2D. An alternative is dimension independent approaches. However, they are still implemented separately for each dimension. The main reason is lack of efficient data structures in the current programming languages. This research goes one step up the ladder of dimension independency in spatial operations. It implements dimension independent spatial operations using the concept of n-simplex. This n-dimensional data type is defined as a list, and its fundamental operations (e.g., dimension, orientation, boundary, clockwise and anticlockwise tests, etc.) are developed as higher order functions over lists, which are treated efficiently in functional programming languages. Some spatial operations (e.g., distance and convex hull computations) have been implemented as case studies. A graphical user interface written with wxHaskell functions has been developed to illustrate the graphical results. The following figure shows the results of the convex hull computation for some 2D and 3D points.



Further reading

- F. Karimipour, M.R. Delavar, and A.U. Frank. A Mathematical Tool to Extend 2D Spatial Operations to Higher Dimensions, Proceedings of the International Conference on Computational Science and Its Applications (ICCSA 2008), (O. Gervasi, B. Murgante, A. Lagan, D. Taniar, Y. Mun, and M. Gavrilova, eds.), Perugia, Italy, June 30 – July 3, 2008, Lecture Notes in Computer Science, Berlin: Springer, Vol. 5072, pp. 153–164.
- F. Karimipour, A.U. Frank, and M.R. Delavar. An Operation-Independent Approach to Extend 2D Spatial Operations to 3D and Moving Objects, Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM GIS 2008), Irvine, CA, USA, November 5–7, 2008.

6.7 Proof Assistants and Reasoning

6.7.1 Galculator

Report by:	Paulo Silva
Status:	unstable, work in progress

The *Galculator* is a prototype of a proof assistant based on the algebra of Galois connections. When combined with the pointfree transform and tactics such as the indirect equality principle, Galois connections offer a very powerful, generic device to tackle the complexity of proofs. The implementation of *Galculator* strongly relies on Generalized Algebraic Data Types (GADTs) which are used in the definition of small Domain Specific Languages. Moreover, they are also used to build an explicit type representation, allowing for a restricted form of dependent type programming.

The prototype of *Galculator* is being developed under an ongoing PhD project. It is still experimental and things tend to change quickly. Since the last report, variables ranging over the universe of types were added to the type representation; a unification mechanism was also defined. This allows us to represent polymorphic proof-objects and having a type inference system to automatically infer their type representations. The details can be found in an article published in *PPDP'08*.

The source code is available from a public SVN repository accessible from the project homepage. After reaching a stable version it will also be available from Hackage.

Currently, we are working on the automatic derivation of the so-called "free-theorems" of polymorphic functions (→3.3.2) and their application to proofs. Moreover, more complex constructions of Galois connections are also being studied. Finally, we plan to integrate the *Galculator* with a theorem prover, namely *Coq*.

Further reading

http://www.di.uminho.pt/research/galculator

6.7.2 funsat: DPLL-style Satisfiability Solver

Report by:	Denis Bueno
Status:	First release imminent, repository available

funsat (mnemonic: functional SAT solver) is a modern satisfiability solver in Haskell, intended to be competitive with state-of-the-art solvers (which are mostly written in C/C++). The strategy is to draw on many ideas from the literature and implement them in a way that is functional, testable, and difficult to accomplish concisely in a lower-level language. Currently the emphasis is on techniques for solving structured, rather than randomized, instances.

Funsat can solve many structured instances from satlib (http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html) including PARITY (16 series), BF, blocksworld, and logistics. Many are solved in a few seconds.

Further reading

The code in its current state is available as a git repository:

```
$ git clone http://churn.ath.cx/funsat
```

6.7.3 sat-micro-hs: SAT-Micro in Haskell

Report by:	Denis Bueno
Status:	Version 0.1.1

Sat-micro-hs is a Haskell port of the OCaml satisfiability (SAT) solver described in "SAT-Micro: petit mais costaud!" ("SAT-Micro: small but strong!", see below). The paper describes a minimal solver with the *flavor* of a modern SAT solver, without the *robustness* necessary for solving hard SAT instances. This port is intended for those interested in SAT generally, as well as any interested in the paper specifically, but who do not read French.

The code is available from Hackage at http://hackage.haskell.org/cgi-bin/hackage-scripts/package/sat-micro-hs.

Further reading

Sylvain Conchon, Johannes Kanig, and Stéphane Lescuyer. "SAT-Micro: petit mais costaud!" In *Dix-neuvièmes Journées Francophones des Langages Applicatifs*, Étretat, France, 2008. INRIA. Available online at http://www.lri.fr/~conchon/publis/conchon-jfla08.ps.

6.7.4 Saoithín: a 2nd-order proof assistant

Report by:	Andrew Butterfield
Status:	ongoing

Saoithín (pronounced "Swee-heen") is a GUI-based 2nd-order predicate logic proof assistant. The motivation for its development is the author's need for support in doing proofs within the so-called "Unifying Theories of Programming" paradigm (UTP). This requires support for 2nd-order logic, equational reasoning, and meets a desire to avoid re-encoding the theorems into some different logical form. It also provides proof transcripts whose style makes it easier to check their correctness.

Saothín is implemented in GHC 6.4 and wxHaskell 0.9.4, with elements of Mark Utting's jaza tool for Z, and has been tested on a range of Windows platforms (98/XP/Vista), and should work in principle on Linux/Mac OS X.

Work has been slow, but help from a summer student intern got the infrastructure for parsing from, and pretty-printing to LaTeX up and going. This will allow complete theories to be entered using LaTeX, and pretty-printing of the resulting proofs. A first public release of the software in some form is now hoped for in early 2009.

Further reading

https://www.cs.tcd.ie/Andrew.Butterfield/Saoithin

6.7.5 Inference Services for Hybrid Logics

Report by:	Guillaume Hoffmann
Participants:	Carlos Areces, Daniel Gorin

"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, available at http://code.google.com/p/intohylo/. Most of them are (successfully) written in Haskell. See HyLoRes (→6.7.6), HTab (→6.7.7), and HGen (→6.7.8).

6.7.6 HyLoRes

Report by:	Guillaume Hoffmann
Participants:	Carlos Areces, Daniel Gorin
Status:	active development
Current release:	2.4

HyLoRes is an automated theorem prover for hybrid logics (→ 6.7.5) 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). We have very recently added support for model-generation for satisfiable formulas.

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 and source: http://code.google.com/p/intohylo/

6.7.7 HTab

Report by:	Guillaume Hoffmann
Participants:	Carlos Areces, Daniel Gorin
Status:	active development
Current release:	1.3.5

HTab is an automated theorem prover for hybrid logics (→ 6.7.5) based on a tableau calculus. It implements a terminating tableau algorithm for the basic hybrid logic extended with the universal modality.

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

Further reading

- Hoffmann, G. and Areces, C. *HTab: a terminating tableaux system for hybrid logic.* In Methods for Modalities 5, Cachan, France, 2007.
- Site and source: http://code.google.com/p/intohylo/

6.7.8 HGen

Report by:	Guillaume Hoffmann
Participants:	Carlos Areces, Daniel Gorin
Status:	active development
Current release:	1.1

HGen is a random CNF (conjunctive normal form) generator of formulas for different hybrid logics. It is highly parameterized to obtain tests of different complexity for the different languages. It has been extensively used in the development of HyLoRes (→ 6.7.6) and HTab (→ 6.7.7).

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.
- Site and source: http://code.google.com/p/intohylo/

6.7.9 Sparkle

Report by:	Maarten de Mol
Participants:	Marko van Eekelen, Rinus Plasmeijer
Status:	stable, maintained

Sparkle is an LCF-style proof assistant dedicated to reasoning about lazy functional programs. It operates on a simplified subset of Clean (→ 3.2.3), and it makes use of the Clean compiler to automatically translate Clean programs to its accepted subset. Sparkle fully supports the semantics of a lazy functional programming language, including lazy evaluation, bottom-values, and manual strictness. The reasoning steps of Sparkle are tailored towards functional programmers, and include both modified ones (such as Reduce and Induction) and unique ones (such as Definedness).

Sparkle is a stand-alone application, written in Clean and with an extensive graphical user interface written in Object I/O. It is only available on Windows platforms.

Further reading

http://www.cs.ru.nl/~Sparkle

6.8 Modeling and Analysis

6.8.1 Coconut

Report by:	Wolfram Kahl
Participants:	Christopher K. Anand
Status:	on-going development

Coconut (COde CONstructing User Tool) is a special-purpose compiler project aiming to provide a coherent tool bench for the development of high-performance, high-assurance scientific computation, and to cover the full range of development activity from mathematical modeling to verification.

Development of the integrated tool bench is in (GHC-)Haskell, and currently proceeding from the bottom up. As code generation target, we are for now focusing on support for the Cell BE, and in particular the special-purpose SPU compute engines of which there are eight on a single Cell BE chip.

A type-indexed embedded DSL for declarative assembly language includes complex SIMD-ization patterns which are easier to encapsulate in the DSL and apply across 30 functions than they would be to implement even for a single function in C. To support rapid prototyping, Coconut includes instruction semantics sufficient to simulate SIMD instruction execution within Haskell. This significantly reduces the time spent on developing new patterns and exploring edge cases for mixed fixed/floating point computations by debugging and unit testing right in GHCi.

The central internal representation are "code graphs", which are used to represent both data-flow graphs and control flow graphs, with separate levels of nesting for non-concurrent and concurrent control flow. For scheduling simple loop bodies programmed in the DSL, we use our Explicitly Staged Software Pipelining (ExSSP) algorithm on the data-flow code graph representation.

Our implementation of single-precision special functions (sin, sinh, asin, ..., sqrt, cbrt, exp, log, lgamma, ...) for the SPU is distributed as MASS in the Cell BE SDK 3.0 in both generated C (for inlining) and long vector functions scheduled by Coconut. In comparison with a state-of-the-art hand-tuned C implementation of these library functions using in-line assembly in the form of intrinsic functions and scheduled by the compiler `spu-xlc`, the Coconut-generated and -scheduled implementations are roughly four times faster; in many cases we know that our implementations are optimal.

We are currently making good progress on novel control flow patterns and scheduling algorithms to support them, a virtual machine model for multicore architectures, and verification strategies for SIMD-ization and multicore synchronization, in an effort to bring the level of usability we have achieved with the DSL for SIMD-ization also to multicore parallelism.

Further reading

http://coconut.mcmaster.ca/

6.8.2 Streaming Component Combinators

Report by:	Blažević Mario
Status:	Experimental, actively developed

Streaming Component Combinators are an experiment at modeling dataflow architecture by using composable streaming components. All components are categorized into a small set of component types. A number of components can be composed into a compound component using a component combinator. For example, two transducer components can be composed together using a *pipe* operator into another transducer; one splitter and two transducers can be composed using an *if* combinator into a single compound transducer. Components are implemented as coroutines, and the data flow among them is synchronous.

There are two ways to use SCC: as an embedded language in Haskell, or as a set of commands in a command-line shell. The latter provides its own parser and type checker, but otherwise relies on the former to do the real work.

The original work was done in the OmniMark programming language. Haskell was the language of choice for the second implementation because its strong typing automatically makes the embedded language strongly typed, and because its purity forces the implementation to expose the underlying semantics.

The currently planned future work includes extending the set of primitive components and component combinators and improving their performance and the scripting abilities of the shell interface.

The latest stable version of SCC is available from Hackage.

Further reading

- Hackage: http://hackage.haskell.org/cgi-bin/hackage-scripts/package/scc-0.2
- Conference paper: Mario Blažević, Streaming component combinators, Extreme Markup Languages, 2006. http://www.idealliance.org/papers/extreme/proceedings/html/2006/Blazevic01/EML2006Blazevic01.html
- OmniMark implementation: http://developers.omnimark.com/etcetera/streaming-component-combinators.tar.gz

6.8.3 Raskell

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

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. 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.

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 (→ 5.5.6), 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⟩

6.8.4 iTasks

Report by:	Thomas van Noort
Participants:	Rinus Plasmeijer, Peter Achten, Pieter Koopman, Bas Lijnse
Status:	active development

The iTask system is a set of combinators to specify workflow in the pure and functional language Clean (→ 3.2.3) at a very high level of abstraction. Workflow systems are automated systems in which tasks are coordinated that have to be executed by either humans or computers. The combinators that are available support workflow patterns commonly found in commercial workflow systems. In addition, we introduce novel workflow patterns that capture real world requirements, but that cannot be dealt with by current systems. For example, work can be interrupted and subsequently directed to other workers for further processing. Compared with contemporary workflow systems, the iTask system offers several further advantages:

- Tasks are statically typed and can be higher-order.
- Combinators are fully compositional.
- Dynamic and recursive workflow is supported.
- An executable web-based multi-user workflow application is generated from a specification.

The iTask system makes extensive use of Clean's generic programming facilities and its iData toolkit with which interactive, thin-client, form-based web applications can be created.

Future plans

Currently, we are working on demand driven workflow specifications in which a task only needs to be completed when its result is required. Furthermore, we are looking at how changes in a specification affect a running workflow application.

Further reading

- http://wiki.clean.cs.ru.nl/ITasks
- http://www.st.cs.ru.nl/Onderzoek/Publicaties/publicaties.html

6.9 Hardware Design

6.9.1 ForSyDe

Report by:	Alfonso Acosta
Participants:	Ingo Sander, Axel Jantsch, Zhonghai Lu, Tarvo Raudvere, Jun Zhu
Status:	Experimental

The ForSyDe (Formal System Design) methodology has been developed with the objective to move system-on-chip design to a higher level of abstraction. ForSyDe is implemented as a Haskell-embedded behavioral DSL.

We have recently released ForSyDe 3.0, which includes a new deep-embedded DSL and embedded compiler with different backends (Simulation, Synthesizable VHDL and GraphML), as well as a new user-friendly tutorial.

The source code, together with example system models, is available from HackageDB under the BSD3 license.

Features

ForSyDe includes two DSL flavors which offer different features:

1. Deep-embedded DSL

   Deep-embedded signals (`ForSyDe.Signal`), based on the same concepts as Lava (→ 6.9.2), are aware of the system structure. Based on that structural information ForSyDe's embedded compiler can perform different analysis and transformations.

   - Thanks to Template Haskell, computations are expressed in Haskell, not needing to specifically design a DSL for that purpose.
   - Embedded compiler backends:
     - Simulation
     - VHDL (with support for Modelsim and Quartus II)
     - GraphML (with yFiles graphical markup support.)
   - Synchronous model of computation.
   - Support for components.
   - Support for fixed-sized vectors.

2. Shallow-embedded DSL

   Shallow-embedded signals (`ForSyDe.Shallow.Signal`) are modeled as streams of data isomorphic to lists. Systems built with them are restricted to simulation. However, shallow-embedded signals provide a rapid-prototyping framework which allows to experiment with heterogeneous models of computation (MoCs).

   - Synchronous MoC.
   - Untimed MoC.
   - Continuous Time MoC.
   - Domain Interfaces allow connecting various subsystems with different timing (domains) regardless of their MoC.

ForSyDe allows to integrate deep-embedded models into shallow-embedded ones. This makes it possible to simulate a synthesizable deep-embedded model together with its environment, which may consist of analog, digital, and software parts. Once the functionality of the deep-embedded model is validated, it can be synthesized to hardware using the VHDL-backend of ForSyDe's embedded compiler.

Further reading

http://www.ict.kth.se/org/ict/ecs/sam/projects/forsyde/www/

6.9.2 Lava

Report by:	Emil Axelsson
Participants:	Koen Claessen, Mary Sheeran, Satnam Singh

Lava is a hardware description library embedded in Haskell. By modeling hardware components as functions from inputs to outputs, Lava allows structural hardware description using standard functional programming techniques. The version developed at Chalmers University (http://www.cs.chalmers.se/~koen/Lava/) has a particular aim to support formal verification in a convenient way. The version developed at Xilinx Inc. (http://raintown.org/lava/) focuses on FPGA core generation, and has been successfully used in real industrial design projects.

Some recent Lava-related work at Chalmers is Mary Sheeran's parallel prefix generators, which use a clever search to find networks with a good balance between speed and low power. The most visible activity on Lava itself over the last years is that the Chalmers version has been made available from Hackage.

Further reading

http://www.cs.chalmers.se/~koen/Lava/

6.9.3 Wired

Report by:	Emil Axelsson
Participants:	Koen Claessen, Mary Sheeran

Wired is an extension to the hardware description library Lava (→ 6.9.2), targeting (not exclusively) semi-custom VLSI design. A particular aim of Wired is to give the designer more control over the routing wires' effects on performance.

The goal is a system with the following features:

1. Convenient circuit description in monadic style.

2. Layout/wiring expressed using optional annotations, allowing incremental specification of physical aspects.
3. Export designs to several formats:
   - Lava (for, e.g., verification)
   - Postscript (visualizing layout and wiring)
   - Design Exchange Format (interface to standard CAD tools for, e.g., fabrication)
4. Accurate, wire-aware timing/power analysis within the system.
5. Support for a few modern cell libraries.
6. Automatic modeling of cell libraries.

We are not very far from this goal. The missing parts are power analysis and support for cell libraries.

Wired is still quite unstable and has not been tested in any larger scale. Realistic testing requires support for realistic cell libraries, something which we are actively working on. The upcoming release will be shipped with the free Nangate 45nm Open Cell Library (http://www.nangate.com), allowing users to play with a cutting-edge VLSI technology without the need for any expensive and complicated CAD tools.

Further reading

http://www.cs.chalmers.se/~emax/wired/

6.9.4 Oread

Report by:	Garrin Kimmell
Participants:	Ed Komp, Perry Alexander
Status:	beta, actively developed

The Computer Systems Design Lab is investigating the use of functional languages in the development of mixed-fabric (hardware and software) embedded systems. To this end, we have developed a language, Oread, and an accompanying toolset, implemented in Haskell. Oread is a strict functional language, combined with monadic message-passing architecture, which allows a system to be compiled to both traditional CPU instruction sets and FPGA hardware. The principal application target for Oread programs is the construction of software-defined radio components.

Oread is available for download at the link provided below. Version 0.1 of Oread was released in November 2008.

Further reading

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

Contact

⟨kimmell@ittc.ku.edu⟩

6.10 Natural Language Processing

6.10.1 GenI

Report by:	Eric Kow

GenI is a surface realizer for Tree Adjoining Grammars. Surface realization 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 realization library, several optimizations, 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.

GenI is available on Hackage, and can be installed via cabal-install. We also have a mailing list at http://websympa.loria.fr/wwsympa/info/geni-users.

Further reading

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

6.10.2 Grammatical Framework

Report by:	Krasimir Angelov
Participants:	Aarne Ranta, Björn Bringert, Håkan Burden

Grammatical Framework (GF) is a programming language for multilingual grammar applications. It can be used as a more powerful alternative to Happy but in fact its main usage is to describe natural language grammars instead of for programming languages. The language itself will look familiar for most Haskell or ML users. It is a dependently typed functional language based on Per Martin-Löf's type theory.

An important objective in the language development was to make it possible to develop modular grammars. The language provides modular system inspired from ML but adapted to the specific requirements in GF. The modules system was exploited to a large extent in the Resource Libraries project. The library provides large linguistically motivated grammars for a number of languages. When the languages are closely related the common parts in the grammar could be shared using the modules system. Currently there are complete grammars for Bulgarian, Danish, English, Finnish, French, German, Interlingua, Italian, Norwegian, Russian, Spanish, and Swedish. Some still incomplete grammars are available for Arabic, Catalan, Latin, Thai, and Hindi/Urdu. On top of these grammars a user with limited linguistic background can build application grammars for a particular domain.

In June 2008 a beta version of GF 3.0 was released. This is a major refactoring of the existing system. The code base is about half in size and makes a clear separation between compiler and runtime system. A Haskell library is provided that allows GF grammars to be easily embedded in the user applications. There is a translator that generates JavaScript code which allows the grammar to be used in web applications as well. The new release also provides new parser algorithm which works faster and is incremental. The incrementality allows the parser to be used for word prediction, i.e., someone could imagine a development environment where the programming language is natural language and the user still can press some key to see the list of words allowed in this position just like it is possible in Eclipse, JBuilder, etc.

Further reading

www.digitalgrammars.com/gf

6.11 Inductive Programming

6.11.1 Inductive Programming

Report by:	Lloyd Allison

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

The project is (i) to use Haskell to examine what are the products of artificial-intelligence (AI)/data mining/machine-learning from a programming point of view, and (ii) to do data analysis with them.

IP 1.2 now contains estimators, from given weighted and unweighted data, to the Poisson and Geometric distributions over non-negative integer variables, and Student's t-Distribution over continuous variables. The new (and the earlier) distributions may be used as components to the learners (estimators) of structured models such as unsupervised classifications (mixture models), classification- (decision-, regression-) trees and other function-models (regressions), mixed Bayesian networks, and segmentation models. A small prototype module of numerical/scientific functions, in Haskell, has been added to IP 1.2, to support the implementation of Student's t-Distribution in the first instance.

I am working on some routines for the analysis of labeled graphs (networks), and on reorganizing the modules slightly to suit Haskell's module system better.

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

Future plans

Planned are continuing extensions, applications to real data-sets, and comparisons against other learners.

Further reading

- http://www.allisons.org/ll/FP/IP/
- http://www.csse.monash.edu.au/~lloyd/tildeFP/II/

6.11.2 IgorII

Report by:	Martin Hofmann
Participants:	Emanuel Kitzelmann, Ute Schmid
Status:	experimental, active development

IgorII is a new method and an implemented prototype for constructing recursive functional programs from a few non-recursive, possibly non-ground, example equations describing a subset of the input/output behavior of a target function to be implemented.

For a simple target function like `reverse` the sole input would be the following, the k smallest w.r.t. the input data type, examples:

```
reverse []      = []
reverse [a]     = [a]
reverse [a,b]   = [b,a]
reverse [a,b,c] = [c,b,a]
```

The result, shown below, computed by IgorII is a recursive definition of `reverse`, where the subfunctions `last` and `init` have been automatically invented by the program.

```
reverse []     = []
reverse (x:xs) = (last (x:xs)):(reverse (init (x:xs))

last [x]       = x
last (x:y:ys)  = last (y:ys)
init [x]       = []
init (x:y:ys)  = x:(init (y:ys))
```

Features

- termination by construction
- handling arbitrary user-defined data types
- utilization of arbitrary background knowledge
- automatic invention of auxiliary functions as subprograms
- learning complex calling relationships (tree- and nested recursion)
- allowing for variables in the example equations
- simultaneous induction of mutually recursive target functions

Current Status and Future Plans

IgorII is currently still implemented in the reflective rewriting based programming and specification language Maude, and is available on the project page. However, it will be ported to Haskell soon.

For the future, an extension to higher-order context is planned, as well as the introduction of further functional constructs (e.g., `let`) and accumulator variables.

Further reading

- http://www.cogsys.wiai.uni-bamberg.de/effalip/
- http://www.inductive-programming.org/

6.12 Others

6.12.1 Bioinformatics tools

Report by:	Ketil Malde

The Haskell bioinformatics library supports working with nucleotide and protein sequences and associated data. File format support includes sequences in Fasta (with associated quality information), TwoBit, and PHD formats, BLAST XML output, and ACE alignment files.

The standard alignment algorithms (and some non-standard ones) are provided, as well as sequence indexing, complexity calculation, protein translation, etc.

The library is considered in development (meaning things will be added, some functionality may not be as complete or well documented as one would wish, and so on), but central parts should be fairly well documented and come with a QuickCheck test and benchmarking suite.

The library abstracts functionality that is used in a handful of applications, including:

- `xsact` — an EST clustering program
- `RBR` — a repeat detector/masker
- `clusc` — a tool for calculating cluster similarity with a bunch of metrics
- `dephd` — a sequence quality assessment tool
- `xml2x` — a BLAST postprocessor and GO annotator

Everything is GPLed and available as Darcs repos, at http://malde.org/~ketil/biohaskell/.

6.12.2 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.

lambdabot 4.0 has been released, and is available from Hackage. Cale Gibbard has also kindly taken over maintenance of the bot.

Further reading

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

6.12.3 Roguestar

Report by:	Christopher Lane Hinson
Status:	early development

Roguestar is a science fiction themed roguelike game written in Haskell. Roguestar uses a client-server model: roguestar-engine is the backend game engine, while roguestar-gl is the OpenGL client.

Roguestar 0.2.2 was announced on 16th August 2008, and is the first version to support Windows.

RSAGL is the RogueStar Animation and Graphics Library, which was written specifically to support roguestar-gl. Every effort has been made to make it accessible to other projects. It includes domain-specific primitives, monads, and arrows for 3D modeling and animation.

Roguestar is licensed under the Affero General Public License. RSAGL is licensed under a permissive license.

In the last few months, a major subproject has been the development of a simple fast special-purpose media ray tracer, which is being used to generate realistic extraterrestrial sky spheres.

Further reading

- http://roguestar.downstairspeople.org
- http://blog.downstairspeople.org

6.12.4 Hpysics

Report by:	Roman Cheplyaka
Status:	experimental

Hpysics is a 3-D physics engine written in Haskell. It started as a Google Summer of Code project mentored by Manuel Chakravarty and Roman Leshchinskiy. The distinctive feature of Hpysics is the use of GHC's Data Parallel Haskell extension to enable fast real-time physics simulation on multi-core processors.

At the moment Hpysics supports polyhedral shapes using VClip algorithm for narrow-phase collision detection. A simple OpenGL visualization is included.

Potential users of the physics engine are welcome to guide further development of the project.

The source code is available from public darcs repository under the BSD license.

Further reading

- ◦ http://haskell.org/haskellwiki/Hpysics
- ◦ An article about Hpysics is to appear in the SoC issue of The Monad Reader.

6.12.5 hledger

Report by:	Simon Michael
Participants:	

hledger is a command-line accounting tool similar to John Wiegley's ledger tool.

The first release has been published on Hackage, and has attracted some interest. It can be used for generating simple balance and transaction reports from a plain-text general ledger. A home page and mail list has also been created.

Immediate plans are to add some more of the most useful features from c++ ledger, so that hledger can be used for day-to-day finances, and to grow the community of contributors.

Further reading

http://joyful.com/hledger

7 Commercial Users

7.1 Well-Typed LLP

Report by:	Ian Lynagh
Participants:	Björn Bringert, Duncan Coutts

Well-Typed is a Haskell services company. We provide commercial support for Haskell as a development platform. We also offer consulting services, contracting, and training. For more information, please take a look at our website or drop us an e-mail at ⟨info@well-typed.com⟩.

The three of us formed the company back in March, with a plan to take care of our existing commitments (the GHC support contract and two PhD theses) and to go full time around the end of the summer. Since then, Björn has accepted a position at Google, although he remains as a partner in an advisory role. Meanwhile, Well-Typed has kept Ian and Duncan busy full-time for the last couple of months, including taking over the role of GHC support engineer. We are pleased to have been able to play a part in getting the recent GHC 6.10.1 release out.

Looking ahead, we have many exciting prospects on the horizon. We expect to have some interesting announcements before the next HCAR edition. Stay tuned!

Further reading

http://www.well-typed.com/

7.2 SeeReason Partners, LLC

Report by:	Clifford Beshers
Participants:	David Fox, Jeremy Shaw

Clifford Beshers, David Fox, and Jeremy Shaw comprise SeeReason Partners, LLC. We develop web services using Haskell to build our applications. We are currently using AJAX techniques, with Javascript code generated from Haskell, deployed using HAppS as a web server. We initially planned working with Adobe Flash, based on early work on a Haskell to Flash compiler, but have postponed that work for now.

We have developed and deployed a website for creating art appraisal reports, in use by a private firm. These documents require specific formatting that must be more flexible than simple boilerplate, but for which standard commercial word processing tools proved to be too cumbersome. Multiple users can edit reports simultaneously through a web interface using Wiki markup, which is converted to LaTeX and rendered in PDF format.

We are currently working towards launching AlgebraZam.com, a site to teach mathematics skills, beginning with elementary algebra. The initial launch, expected fourth quarter 2008, will begin with an interactive tool for solving simple algebraic equations.

Formerly core members of the operating systems group at Linspire, Inc., we continue to maintain the tools for managing a Debian Linux distribution that we developed there. Source code for these tools can be found at our public source code repository http://src.seereason.com/. These include a package build system (`autobuilder`) as well as Cabal to Debian conversion tool (`cabal-debian`). We provide current archives of many Haskell packages (including GHC 6.8.3 built with Haddock 2.x) built for recent versions of Debian (unstable) and Ubuntu (8.04 and soon 8.10.) Packages are available at http://deb.seereason.com/. We welcome inquiries from developers interested in using these packages or helping out with continued development.

We can be reached at ⟨(cliff,david,jeremy)@seereason.com⟩ and on `#haskell` (→ 1.2) respectively as thetallguy, dsfox, and stepcut.

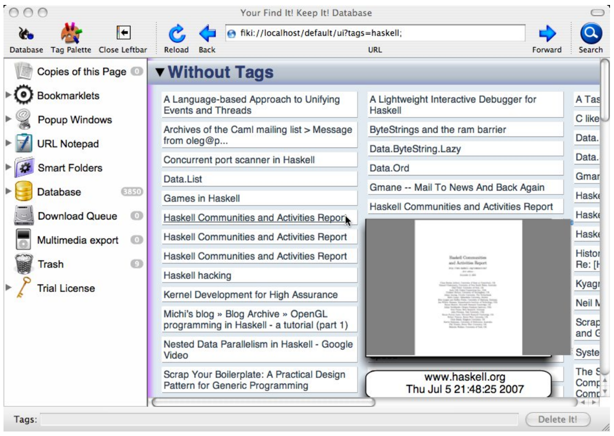
7.3 Ansemond LLC

Report by:	Sengan Baring-Gould

Find It! Keep It! is a Mac Web Browser that lets you keep the pages you visit in a database. A list of these pages is shown in the "database view". This view is rendered by the browser from generated HTML and is dynamically updated by Javascript DOM operations: tens of thousands of elements cannot efficiently be placed on the screen using DOM operations only, while rerendering a half a megabyte of HTML each time a user interface element changes is unresponsive. A glitch free user experience requires keeping these two separate mechanisms synchronized, which proved difficult.

A Haskell implementation generates abstract DOM operations which are then either rendered to HTML or are converted to Javascript DOM operations to be run within the browser. While this process is not complex (difficult algorithm) it is complicated (hard for the programmer to keep everything in mind). The additional modularity afforded by laziness proved invaluable, enabling all the different pieces to be coded much more independently and clearly than was possible in the original Python version. This same engine could be used on a web server and would work with any web browser. The Haskell version is scheduled to ship in version 1.1

of *Find It! Keep It!*


Your Find It! Keep It! Database
Database
Tag Palette
Close Leftbar
Reload
Back
fiki://localhost/default/ui?tags=haskell;
URL
Forward
Search
Copies of this Page
Bookmarklets
Popup Windows
URL Notepad
Smart Folders
Database
Download Queue
Multimedia export
Trash
Trial License
Without Tags
A Language-based Approach to Unifying Events and Threads
Archives of the Caml mailing list > Message from oleg@p...
Concurrent port scanner in Haskell
Data.List
Games in Haskell
Haskell Communities and Activities Report
Haskell Communities and Activities Report
Haskell Communities and Activities Report
Haskell hacking
Kernel Development for High Assurance
Michi's blog » Blog Archive » OpenGL programming in Haskell - a tutorial (part 1)
Nested Data Parallelism in Haskell - Google Video
Scrap Your Boilerplate: A Practical Design Pattern for Generic Programming
A Lightweight Interactive Debugger for Haskell
ByteStrings and the ram barrier
Data.ByteString.Lazy
Data.Ord
Gmane -- Mail To News And Back Again
Haskell Communities and Activities Report
www.haskell.org
Thu Jul 5 21:48:25 2007
Tags:
Delete It!

Further reading

http://www.ansemond.com

7.4 Credit Suisse Global Modeling and Analytics Group

Report by:	Ganesh Sittampalam

GMAG, the quantitative modeling 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 modelers 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 modelers 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 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.

Our past projects include:

- Adding higher-order functions to Excel, implemented via (Haskell) addin code.
- Tools to transform spreadsheets into directly executable code.
- A "lint" tool to check for common errors in spreadsheets.

Our main current project is Paradise, a domain-specific language embedded in Haskell for implementing reusable components that can be compiled into multiple target forms. It has been under development for a long time now, and over that time the team working on it has grown to several people.

Paradise's first target form was Excel spreadsheets, and that backend is now relatively mature; our main focus at the moment is generating .NET components which can be run standalone or plugged into a bank-wide system. In future we may target yet more diverse platforms such as web browsers.

Several modelers have now been exposed directly to Haskell by using Paradise, and they have generally picked it up fairly quickly. All new recruits are now introduced to Haskell as part of our internal training program.

Further reading

- CUFP 2006 talk about Credit Suisse:
  http://cufp.galois.com/slides/2006/HowardMansell.pdf
- ICFP 2008 experience report about Paradise:
  http://www.earth.li/~ganesh/research/paradise-icfp08/paper.pdf
  http://www.earth.li/~ganesh/research/paradise-icfp08/talk.pdf

7.5 Bluespec tools for design of complex chips

Report by:	Rishiyur Nikhil
Status:	Commercial product

Bluespec, Inc. provides a language, BSV, which is being used for all aspects of ASIC and FPGA system design — specification, synthesis, modeling, and verification. All hardware behavior is expressed using *rewrite rules* (Guarded Atomic Actions). BSV borrows many ideas from Haskell — algebraic types, polymorphism, type classes (overloading), and higher-order functions. Strong static checking extends into correct expression of multiple clock domains, and to gated clocks for power management. BSV is universal, accommodating the diverse range of blocks found in SoCs, from algorithmic "datapath" blocks to complex control blocks such as processors, DMAs, interconnects and caches.

Bluespec's core tool synthesizes (compiles) BSV into high-quality RTL (Verilog), which can be further synthesized into netlists for ASICs and FPGAs using other commercial tools. Automatic synthesis from atomic transactions enables design-by-refinement, where an initial executable approximate design is systematically transformed into a quality implementation by successively adding functionality and architectural detail. Other products include fast BSV simulation and development tools. Bluespec also uses Haskell to implement its tools (over 100K lines of Haskell).

This industrial strength tool has enabled some large designs (over a million gates) and significant architecture research projects in academia and industry. This kind of research was previously feasible only in software simulation. BSV permits the same convenience of expression as SW languages, and its synthesizability further allows execution on FPGA platforms at three orders of magnitude greater speeds, making it possible now to study realistic scenarios.

Status and availability

BSV tools, available since 2004, are in use by several major semiconductor companies and universities. The tools are free for academic teaching and research.

Further reading

- R.S.Nikhil, *Bluespec, a General-Purpose Approach to High-Level Synthesis Based on Parallel Atomic Transactions*, in *High Level Synthesis: from Algorithm to Digital Circuit, Philippe Coussy and Adam Morawiec (editors)*, Springer, 2008, pp. 129-146.
- Small illustrative examples: http://www.bluespec.com/wiki/SmallExamples
- Winning entry in MEMOCODE 2008 design contest: http://rijndael.ece.vt.edu/memocontest08/
- MIT courseware, "Complex Digital Systems": http://csg.csail.mit.edu/6.375
- A fun example with many functional-programming features — BluDACu, a parameterized Bluespec hardware implementation of Sudoku: http://www.bluespec.com/products/BluDACu.htm

7.6 Galois, Inc.

Report by:	Andy Adams-Moran

Galois 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.functionalprogramming.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, including FPGAs); a GUI debugger for a specialized microprocessor; a specialized, high assurance, cross-domain web and file server, 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.

Web-based technologies are increasingly important to our clients, and we believe Haskell has a key role to play in the production of reliable, secure web software. The culture of correctness Haskell encourages is ideally suited to web programming, where issues of security, authentication, privacy, and protection of resources abound. In particular, Haskell's type system makes possible strong static guarantees about access to resources, critical to building reliable web applications.

To help push further the adoption of Haskell in the domain of web programming, Galois released a suite of Haskell libraries, including:

- json: Support for JavaScript Object Notation
- xml: A simple, lightweight XML parser/generator.
- utf8-string: A UTF8 layer for IO and Strings.
- selenium: Communicate with a Selenium Remote Control server.
- curl: libcurl is a rich client-side URL transfer library.
- sqlite: Haskell binding to sqlite3 databases.
- feed: Interfacing with RSS and Atom feeds
- mime: Haskell support for working with MIME types.

Continuing our deep involvement in the Haskell community, Galois was happy to sponsor the two Haskell hackathons held in the past year, Hac 07 II, in Freiburg, Germany, and Hac4 in Gothenburg, Sweden. Galois also sponsored the second BarCamp Portland, held in early May 2008.

Further reading

http://www.galois.com/.

7.7 IVU Traffic Technologies AG Rostering Group

Report by:	Michael Marte
Status:	Released

The rostering group at IVU Traffic Technologies AG has been using Haskell to check rosters for compliance with the "EC Regulation No 561/2006 on the harmonization of certain social legislation relating to road transport" which "lays down rules on driving times, breaks and rest periods for drivers engaged in the carriage of goods and passengers by road".

By reduction from SEQUENCING WITH RELEASE TIMES AND DEADLINES (Garey & Johnson, Computers & Tractability, 1977), it is easy to show that EC 561/2006 is NP complete due to combinatorial rest-time compensation rules.

Our implementation is based on an embedded DSL to combine the regulation's single rules into a solver that not only decides on instances but, in the case of a faulty roster, finds an interpretation of the roster that is "favorable" in the sense that the error messages it entails are "helpful" in leading the dispatcher to the resolution of the issue at hand.

Our EC 561/2006 solver comprises about 1700 lines of Haskell code (including about 250 lines for the C API), is compiled to a DLL with ghc, and linked dynamically into C++ and Java applications. The solver is both reliable (due to strong static typing and referential transparency — we have not experienced a failure in three years) and efficient (due to constraint propagation, a custom search strategy, and lazy evaluation).

Our EC 561/2006 component is part of the IVU.crew software suite and as such is in wide-spread use all over Europe, both in planning and dispatch. So the next time you enter a regional bus, chances are that the driver's roster was checked by Haskell.

Further reading

- EC 561/2006 at EurLex
- The IVU.suite for public transport

7.8 Tupil

Report by:	Chris Eidhof
Participants:	Eelco Lempsink

(,) tupil

Tupil builds reliable web software with Haskell. Using Haskell's powerful ways of abstraction, we can develop with the speed of dynamic scripting languages but with the safety and performance of a language that is statically checked and compiled.

Because we like to give back to the community, we have released a formlets library that is based on the formlets by Wadler et al. Also, we have implemented the protocol for a Sphinx client. This allows you to do fast full-text searching using Haskell. Of course, we plan to release more libraries in the future.

Further reading

- http://tupil.com
- http://blog.tupil.com

8 Research and User Groups

8.1 Functional Programming Lab at the University of Nottingham

Report by: Liyang HU

The School of Computer Science at the University of Nottingham has recently formed the *Functional Programming Laboratory*, a new research group focused on all theoretical and practical aspects of functional programming, together with related topics such as type theory, category theory, and quantum programming.

The laboratory is led jointly by Thorsten Altenkirch and Graham Hutton, with Henrik Nilsson and Venanzio Capretta as academic staff. With 4 more research staff and some 10 PhD students in our group, we have a wide spectrum of interests:

Containers

Nottingham has been home to the EPSRC grant on *containers*, a semantic model of functional data structures. Thorsten Altenkirch, Peter Hancock, Peter Morris, and Rawle Prince are working with containers to both write and reason about programs. Peter Morris has recently finished his PhD, which used containers as a basis for generic programming with dependent types.

Dependently Typed Programming (DTP)

Peter Morris and Nicolas Oury are working on Epigram, while Nils Anders Danielsson is involved in the development of Agda (→ 3.2.2). Our interests lie both in the pragmatics of using DTP, as witnessed by work on libraries and tools, and in foundational theory, including the Observational Type Theory underlying Epigram 2 and James Chapman's work on normalization. DTP is also used to control and reason about effects, and a number of us are using Agda as a proof assistant to verify programs or programming language theory.

Functional Reactive Programming (FRP)

The FRP team are working on FRP-like and FRP-inspired declarative, domain-specific languages. Under Henrik Nilsson's supervision, Neil Sculthorpe is working on a new, scalable FRP language based on reactive components with multiple inputs and outputs, while George Giorgidze is applying the advantages of FRP to non-causal modeling with the aim of designing a new, more expressive and flexible language for non-causal, hybrid modeling and simulation (→ 6.5.3). Tom Nielsen is implementing a declarative language for experiments, simulations, and analysis in neuroscience. A theme that permeates our work is implementation through embedding in typed functional languages such as Haskell or Agda (→ 3.2.2). The team also maintains Yampa, the latest Haskell-based implementation of FRP.

Quantum Programming

Thorsten Altenkirch and Alexander S Green have been working on the Quantum IO Monad (QIO), an interface from Haskell to Quantum Programming. Taking advantage of abstractions available in Haskell we can provide QIO implementations of many well-known quantum algorithms, including Shor's factorization algorithm. The implementation also provides a constructive semantics of quantum programming in the form of a simulator for such QIO computations.

Reasoning About Effects

Graham Hutton and Andy Gill recently formalized the worker/wrapper transformation for improving the performance of functional programs. Wouter Swierstra and Thorsten Altenkirch have produced functional specifications of the IO monad, as described in Wouter's forthcoming PhD thesis. Mauro Jaskelioff developed a new monad transformer library for Haskell, which provides a uniform approach to lifting operations. Diana Fulger and Graham Hutton are investigating equational reasoning about various forms of effectful programs. Liyang HU and Graham Hutton are working on verifying the correctness of compilers for concurrent functional languages, including a model implementation of software transactional memory.

Teaching

Haskell plays an important role in the undergraduate program at Nottingham, as well as our China and Malaysia campuses. Modules on offer include Functional Programming, Advanced FP, Mathematics for CS, Foundations of Programming, Compilers, and Computer-Aided Formal Verification, among others.

Events

The FP Lab plays a leading role in the Midlands Graduate School in the Foundations of Computing Science, the British Colloquium for Theoretical Computer Science, and the Fun in the Afternoon seminar series on functional programming.

FP Lunch

Every Friday, we gather for lunch with helpings of informal, impromptu-style whiteboard discussions on recent developments, problems, or projects. Summaries of our weekly meetings can be found on the frequently cited *FP Lunch blog*, giving a lively picture of ongoing research at Nottingham.

Later in the afternoon, there is usually a formal hour-long seminar. We are always keen on speakers in any related areas: do get in touch with Thorsten Altenkirch ⟨txa@cs.nott.ac.uk⟩ if you would like to visit. See you there!

8.2 Artificial Intelligence and Software Technology at Goethe-University Frankfurt

Report by:	David Sabel
Participants:	Manfred Schmidt-Schauß

An ongoing research topic of the group in Frankfurt is program semantics and analyses of programs.

Deterministic calculi. We proved correctness of strictness analysis using abstract reduction, which was implemented by Nöcker for Clean (→ 3.2.3), and by Schütz for Haskell. Our proof is based on the operational semantics of an extended call-by-need lambda calculus which models a core language of Haskell. Furthermore, we proved equivalence of the call-by-name and call-by-need semantics of an extended lambda calculus with `letrec`, `case`, and constructors. This is of practical relevance, since the semantics of Haskell is call-by-name but nevertheless almost all implementations of Haskell use call-by-need evaluation.

Nondeterministic calculi. We explored several nondeterministic extensions of call-by-need lambda calculi and their applications. We analyzed a model for a lazy functional language with direct-call I/O providing a semantics for `unsafePerformIO`-calls in Haskell. We investigated a call-by-need lambda-calculus extended by parallel-or and its applications as a hardware description language. Most recently, we analyzed a call-by-need lambda calculus extended with McCarthy's `amb` and an abstract machine for lazy evaluation of concurrent computations. For all these investigations an emphasis of our research lies in proving program equivalences based on contextual equivalence for showing correctness of program transformations.

Simulation-based proof techniques. We have shown that the soundness proof (w.r.t. contextual equivalence) for mutual similarity of *Matthias Mann* scales up to a class of untyped higher-order non-deterministic call-by-need lambda calculi. For non-deterministic call-by-need calculi with `letrec`, known approaches to prove such a result are inapplicable. Recently, in collaboration with *Elena Machkasova* we obtained correctness of a variation of simulation for checking contextual equivalence in an extended non-deterministic call-by-need lambda-calculus with `letrec`. Ongoing research is to adapt and extend the methods to an appropriately defined simulation, and to investigate an extension of the methods to a combination of may- and must-convergence.

Concurrency primitives. Most recently, we analyzed the expressivity of concurrency primitives in different functional languages. In collaboration with *Jan Schwinghammer* and *Joachim Niehren*, we showed how to encode Haskell's MVars into a lambda calculus with storage and futures which is an idealized core language of Alice ML. We proved correctness of the encoding using operational semantics and the notions of adequacy and full-abstractness of translations.

Further reading

http://www.ki.informatik.uni-frankfurt.de/research/HCAR.html

8.3 Functional Programming at the University of Kent

Report by:	Olaf Chitil

We are a group of staff and students with shared interests in functional programming. While our work is not limited to Haskell — in particular our interest in Erlang has been growing — Haskell provides a major focus and common language for teaching and research. We are seeking PhD students for funded research projects.

Our members pursue a variety of Haskell-related projects, some of which are reported in other sections of this report. Both Chris Brown and Thomas Davie submitted this September their theses on refactoring and tracing, respectively. Two new students are joining this year. Over the summer Ivan Ivanovski, an IAESTE student from Macedonia, worked with Olaf Chitil on improving Heat (→ 4.4.4) and making it ready for a public release. Heat is a deliberately simple IDE for teaching Haskell that has been used at Kent for three years. 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. The Kent Systems Research Group is developing an occam compiler in Haskell (Tock). Neil Brown has created a Haskell library ("Communicating Haskell Processes") based on the Communicating Sequential Processes calculus.

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/
- Tracing and debugging with Hat: http://www.haskell.org/hat
- Heat: http://www.cs.kent.ac.uk/projects/heat/
- Vital: http://www.cs.kent.ac.uk/projects/vital/
- Pivotal: http://www.cs.kent.ac.uk/projects/pivotal/
- Tock: https://www.cs.kent.ac.uk/research/groups/sys/wiki/Tock
- CHP http://www.cs.kent.ac.uk/projects/ofa/chp/

8.4 Foundations and Methods Group at Trinity College Dublin

Report by:	Andrew Butterfield
Participants:	Glenn Strong, Hugh Gibbons, Edsko de Vries

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

- Supporting OO-Design technique for functional programmers
- Using functional programs as invariants in imperative programming
- Developing a GUI-based 2nd-order equational theorem prover (→ 6.7.4)
- New approaches to uniqueness typing, applicable to Hindley-Milner style type-inferencing (→ 3.3.1)
- Equational reasoning for Concurrent Haskell (new)

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.

Further reading

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

8.5 Formal Methods at DFKI Bremen and University of Bremen

Report by:	Christian Maeder
Participants:	Mihai Codescu, Dominik Lücke, Christoph Lüth, Christian Maeder, Till Mossakowski, Lutz Schröder

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

We are using the Glasgow Haskell Compiler and many of its extensions to develop the Heterogeneous tool set (Hets). Hets consists of parsers, static analyzers, and proof tools for languages from the CASL family, such as CASL itself, HasCASL, CoCASL, CspCASL, and ModalCASL, and additionally OMDoc and Haskell (via Programatica). The Hets implementation is also based on some old Haskell sources such as bindings to uDrawGraph (formerly Davinci) and Tcl/TK that we maintain.

HasCASL is a general-purpose higher order language which is in particular suited for the 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.

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 successor 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.

The Coalgebraic Logic Satisfiability Solver CoLoSS is being implemented jointly at DFKI Bremen and at the Department of Computing, Imperial College London. The tool is generic over representations of the syntax and semantics of certain modal logics; it uses the Haskell class mechanism, including multi-parameter type classes with functional dependencies, extensively to handle the generic aspects.

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 http://www.informatik.uni-bremen.de/htk/ http://www.informatik.uni-bremen.de/uDrawGraph/
- Proof General Kit: http://proofgeneral.inf.ed.ac.uk/Kit
- The Coalgebraic Logic Satisfiability Solver CoLoSS: http://www.informatik.uni-bremen.de/~lschrode/projects/GenMod, http://www.doc.ic.ac.uk/~dirk/COLOSS/

8.6 SCIence project

Report by:	Kevin Hammond
Status:	Ongoing 5-year project, started in 2006

SCIEnce is a 3.2M euros project that brings together major developers of symbolic computing systems, including GAP, KANT, Maple, and MuPAD, and with the world-leading Centre for Research in Symbolic Computation at RISC-Linz (Austria), OpenMath experts from the Technical University of Eindhoven (Netherlands), and functional programming experts in the Heriot-Watt University (Edinburgh, Scotland) and the University of St Andrews (Scotland).

Our research activity — *"Symbolic computing on the Grid"* — 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.

Our results in this direction are reflected in more than 30 publications and a number of research talks and presentations, listed on the project's website. The public downloads are now under revision and the updated version should appear soon.

Further reading

http://www.symbolic-computation.org/

8.7 Functional Programming at K.U.Leuven, Belgium

Report by:	Tom Schrijvers
Participants:	Pieter Wuille

We are a two-man unit of functional programming research within the Declarative Languages and Artificial Intelligence group at the Katholieke Universiteit Leuven, Belgium.

Tom Schrijvers is pursuing a number of topics in Haskell related to Constraint Programming.

◦ *Type Checking:* Recent results are on type checking for type families, and on confluence for non-full functional dependencies. Ongoing work concerns the simplification of type checking for Haskell's extensive type system, and adding new extensions. This is joint work with Martin Sulzmann, Simon Peyton Jones, and Manuel Chakravarty.

◦ *Test Generation:* Together with master student Timmy Weytjens I am looking at constraint-based generation of test cases. The constraint-based approach should be a more efficient than related random (QuickCheck), exhaustive (SmallCheck) or semi/pseudo-constraint-based (LazySmallCheck / narrowing-based) approaches for certain applications.

◦ *First-Class Constraint Programming:* The aim of this project is to model the generic aspects of Constraint Programming in Haskell. Of particular interest is the solver-independent framework for compositional search strategies. This project involves also Peter Stuckey and Phil Wadler.

Pieter Wuille recently started his Ph.D., under the supervision of Bart Demoen and Tom Schrijvers, on the automatic transformation from pure to impure data structures to improve complexity. By combining abstraction of data structures, selection of representation and corresponding implementation, and specific compiler-optimizations, we hope to overcome the performance penalty caused by purity. More information can be found in the paper presented at IFL'08. Current work includes implementing a framework to test the required transformations on a simple strict first-order functional language.

Further reading

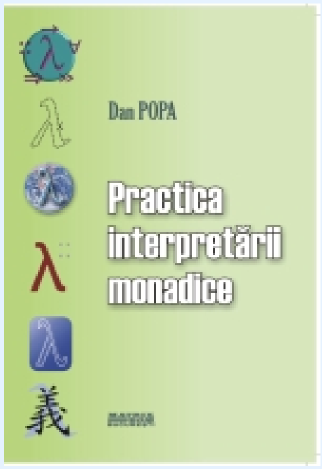
◦ http://www.cs.kuleuven.be/~toms/Haskell/
◦ https://www.cs.kuleuven.be/~pieterw/site/Research/Papers/

8.8 Haskell in Romania

Report by:	Dan Popa

This is to report some activities of the Ro/Haskell group. The Ro/Haskell page becomes more and more known. The numbers of students and teachers interested in Haskell is increasing.

A new book, "The Practice Of Monadic Interpretation" by Dan Popa appears in November 2008.



The book has a nice foreword written by Simon P.J. and is sharing the experience of a year of interpreter building (2006). It is intended as a student's and teacher's guide to the practical approach of monadic interpretation. The book will probably be used during this academic year in 2–4 universities across Romania (in Iasi, Bacau, Cluj-Napoca).

Haskell products like Rodin (a small DSL a bit like C but written in Romanian) begin to spread, proving the power of the Haskell language. The Pseudocode Language Rodin is used as a tool for teaching basics of computer science in some high-schools from various cities. Some teachers from a high school have requested training concerning Rodin.

A group of researchers from the field of linguistics located at the State Univ. from Bacau (The LOGOS Group) is declaring the intention of bridging the gap between semiotics, high level linguistics, structuralism, nonverbal communication, dance semiotics (and some other intercultural subjects) AND Computational Linguistics (meaning Pragmatics, Semantics, Syntax, Lexicology, etc.) using Haskell as a tool for real projects. Probably the situation from Romania is not well known: Romania is probably one of those countries where computational linguistics is studied by computer scientists less than linguists.

At Bacau State University, we will probably teach Haskell on both Faculties: Sciences (The Computers Science being included) and we hope we will work with Haskell with the TI students from the Fac. Of Engineering, where a course on Formal Languages was requested. "An Introduction to Haskell by Examples" had traveled to The Transilvania Univ. (Brasov) and we are expecting Haskell to be used there, too, during this academic year. Other libraries had received manuals and even donations (in books, of course). Editors seem to be interested by the Ro/Haskell movement, and some of them have already declared the intention of helping us by investing capital in the Haskell books production. A well known Publishing House (MatrixRom) asked us to be the Official Publishing House of the Ro/haskell Group.

There are some unsolved problems, too: PhD. Advisors (specialized in monads, languages engineering, and Haskell) are almost impossible to find. This fact seems to block somehow the hiring of good specialists in Haskell. There was even a funny case when somebody hired to teach Haskell was tested and interviewed by a LISP teacher. Of course, the exam was more or less about lists.

Further reading

- Ro/Haskell: http://www.haskell.org/haskellwiki/Ro/Haskell
- Rodin: http://www.haskell.org/haskellwiki/Rodin

8.9 Assorted Small Portland State University Haskell Bits

Report by:	Bart Massey
Participants:	Julian Kongslie, Jamey Sharp
Status:	Mostly under development

Portland State University is a center of development of things like GHC, the House Haskell operating system, etc. Some really amazing work has been done by that group. That group is not us.

Over the past 1.5 years my students and I have participated in a number of Haskell related coding and community activities. Our goals include:

- Better learning and understanding the Haskell programming language.
- Supporting the project work of myself and my group.
- Contributing back to the Haskell community.
- Fun.

Specifically, we have:

- Taken over hosting for the Haskell Sequence / Haskell Weekly News. After some serious initial hiccups, that all seems to be going smoothly.
- Constructed several interesting standalone Haskell applications.
  - Jamey Sharp several years ago constructed a Haskell 802.11 implementation for the Ettus USRP as a Google Summer of Code project.
  - Julian Kongslie has recently completed the first revision of a novel Haskell embedded DSL for Hardware Description, Chortl.
  - Julian Kongslie has recently completed the first revision of a Seaside-style Haskell CGI session framework for web hosting, Riviera.
  - I have mostly completed a Haskell spelling-word suggestion program, thimk, utilizing phonetic codes and edit distance.
- Designed Haskell-supporting hardware. Julian Kongslie has written Verilog implementations of the Haskell G-machine. These are not currently available, as he continues to develop them.
- Constructed library code, and contributed some of it to Hackage and to the Haskell libraries.
  - I have written a command-line argument parsing library, ParseArgs, that I have used in a number of programs. This is in Hackage. However, plans are in the works to convert one of my students' alternate designs currently implemented in C to Haskell, and contribute that instead.

  - I have written a WAVE audio file processing library, and some simple command-line programs for audio processing. These are mostly in Hackage, although I need to package a new release.
  - I have written a PBM graphics file processing library. I am currently working on extending it to support PNG, at which time I will contribute it to Hackage.
  - We have looked at taking over the HaskellDSP signal processing library, after receiving permission from the author to do so. However, this work is currently stalled.
  - In association with my spelling word suggestion application thimk mentioned earlier, I am developing a library of phonetic coding algorithms for contribution to Hackage.
  - I have contributed one patch to the Haskell libraries, to enforce strictness on some sequence constructors as required by the Haskell Report. I also should shortly get my nubOrd function into the libraries.

- Support Haskell in the classroom.
  - Perhaps 20% of my students' projects over the last few years have been submitted in Haskell.
  - I am currently working on tutorial curriculum based on *Real-World Haskell* and the Project Euler problems. Mehana Bisquera-Chang is piloting this curriculum.

- Participated in our local Haskell and open source community. I have given Haskell tutorials at several open source events, such as Portland Bar Camp 2008. Julian and I have both presented our work on several occasions at meetings of the PDX Functional Programming Users' Group.

We would like to thank all of those who have assisted us in this work, but it's hard to enumerate such a large list. Josh Triplett has helped with a number of these projects, and helped me to learn Haskell. Mark Jones and other Faculty at PSU have occasionally helped us out. Don Stewart and others in the Haskell community have provided guidance and support.

What we have been doing is pretty *ad hoc* and disorganized, but we think we have been making real contributions to the Haskell community. Almost all of our work is available to the public under open source licenses — we welcome its use and help with its development.

Further reading

http://wiki.cs.pdx.edu/bartforge