Haskell Communities and Activities Report

http://tinyurl.com/haskcar

Twentieth Edition — May 2011

Janis Voigtländer (ed.)

Andreas Abel
Iain Alexander
Krasimir Angelov
Heinrich Apfelmus
Dmitry Astapov
Christiaan Baaij
Justin Bailey
Alexander Bau
Doug Beardsley
Jean-Philippe Bernardy
Annette Bieniusa
Mario Blažević
Anthonin Bonnefoy
Gwern Branwen
Joachim Breitner
Matt Brown
Björn Buckwalter
Bryan Buecking
Joel Burget
Douglas Burke
Carlos Camarão
Erik de Castro Lopo
Roman Cheplyaka
Olaf Chitil
Duncan Coutts
Nils Anders Danielsson
Dominique Devriese
Daniel Díaz
Atze Dijkstra
Péter Diviánszky
Facundo Dominguez
Marc Fontaine
Patai Gergely
Jürgen Giesl
Brett G. Giles
Andy Gill
George Giorgidze
Dmitry Golubovsky
Marco Gontijo
Matthew Gruen
Torsten Grust
Jurriaan Hage
Sönke Hahn
Malte Harder
Bastiaan Heeren
Judah Jacobson
PÁLI Gábor János
Jeroen Janssen
Csaba Hruska
Oleg Kiselyov
Michal Konečný
Eric Kow
Ben Lippmeier
Andres Löh
Hans-Wolfgang Loidl
Tom Lokhorst
Rita Loogen
Ian Lynagh
John MacFarlane
Christian Maeder
José Pedro Magalhães
Ketil Malde
Alex McLean
Vivian McPhail
Simon Michael
Arie Middelkoop
Neil Mitchell
Dino Morelli
JP Moresmau
Matthew Naylor
Victor Nazarov
Jürgen Nicklisch-Franken
Rishiyur Nikhil
Thomas van Noort
Johan Nordlander
Miguel Pagano
David M. Peixotto
Jens Petersen
Simon Peyton Jones
Dan Popa
Bernie Pope
Antonio M. Quispe
Alberto Ruiz
David Sabel
Antti Salonen
Ingo Sander
Uwe Schmidt
Martijn Schrage
Tom Schrijvers
Jeremy Shaw
Axel Simon
Jan Šnajder
Michael Snoyman
Will Sonnex
Andy Stewart
Martin Sulzmann
Doaitse Swierstra
Henning Thielemann
Simon Thompson
Sergei Trofimovich
Thomas Tuegel
Marcos Viera
Janis Voigtländer
David Waern
Greg Weber
Gregory D. Weber
Kazu Yamamoto
Brent Yorgey

Preface

This is the 20th edition of the Haskell Communities and Activities Report. As usual, fresh entries are formatted using a blue background, while updated entries have a header with a blue background. Entries for which I received a liveness ping, but which have seen no essential update for a while, have been replaced with online pointers to previous versions. Other entries on which no new activity has been reported for a year or longer have been dropped completely. Please do revive such entries next time if you do have news on them.

A call for new entries and updates to existing ones will be issued on the usual mailing lists in October. Now enjoy the current report and see what other Haskellers have been up to lately. Any feedback is very welcome.

Janis Voigtländer, University of Bonn, Germany, ⟨hcar@haskell.org⟩

Contents

1 Community, Articles/Tutorials

1.1 Haskellers

Report by:	Michael Snoyman
Status:	experimental

Haskellers is a site designed to promote Haskell as a language for use in the real world by being a central meeting place for the myriad talented Haskell developers out there. It allows users to create profiles complete with skill sets and packages authored and gives employers a central place to find Haskell professionals.

Since the last HCAR, Haskellers has added job postings, strike forces, and the ever important bling, as well as a brand new, community-developed site design. Haskellers is quickly approaching 800 active accounts. To be clear, the site is intended for all members of the Haskell community, from professionals with 15 years experience to people just getting into the language.

Further reading

http://www.haskellers.com/

1.2 Haskell Wikibook

Participants:	Heinrich Apfelmus, Duplode, Orzetto, David House, Eric Kow, and other contributors
Status:	active development

See: http://www.haskell.org/communities/11-2010/html/report.html#sect1.2.

1.3 The Monad.Reader

Report by:	Brent Yorgey

There are many academic papers about Haskell and many informative pages on the HaskellWiki. 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 might 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 the last HCAR there have been two new issues. Issue 17, published in January 2011, featured articles on difference lists, a new abstraction for interleaving streams of behavior, and abstraction elimination. March saw the publication of a special poetry and fiction edition full of Haskell-related song lyrics, poems, and stories. Issue 18 is planned for release in May 2011.

Further reading

http://themonadreader.wordpress.com/

1.4 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 three additions:

A non-traditional tutorial on Hindley-Milner type inference

This lecture course, developed together with Chung-chieh Shan and presented at the Formosan Summer School on Logic, Language, and Computation (Taipei, Taiwan, July 9-10 2008) teaches writing evaluators, type checkers, type reconstructors and inferencers for a higher-order language with Hindley-Milner type system.

The course is built around the idea that type checking is evaluation with “abstract values”. The course presents, among others, the less-known simple-type inference algorithm that reconstructs not only types but also the type environment, letting us type check open terms and determine environments in which they may be used.

The course explores the deep relation between parametric polymorphism and “inlining”. Polymorphic type checking then is an optimization to type check a polymorphic term at the place of its definition rather than at the places of its use.

◦ http://okmij.org/ftp/Computation/index.html#teval
◦ http://okmij.org/ftp/Computation/FLOLAC/lecture.pdf

Pure functional, mutation-free, efficient double-linked lists

We show a simple example of achieving all the benefits of an imperative data structure — including sharing and efficient updates — in a pure functional program. Our data structure is a doubly-linked, possibly cyclic list, with the standard operations of adding, deleting and updating elements; traversing the list in both directions; and iterating over the list with cycle detection. The code is purely functional, performing no destructive updates, employing no mutable variables such as `IORef`, and using no state monads. Therefore, updates can be easily undone and redone. The code uniformly handles both cyclic and terminated lists. Updating an element takes time bound by a small constant; the update does *not* rebuild the whole list.

It is not for nothing that Haskell has been called the best imperative language. One can implement imperative algorithms just as they are — yet genuinely functionally, without resorting to the monadic sub-language but taking the full advantage of clausal definitions, pattern guards and laziness.

http://okmij.org/ftp/Algorithms.html#pure-cyclic-list

Simple and reliable uni- and bi-directional pipes

MySysOpen, Haskell binding to `sys_open.c`, lets Haskell code interact with another local or remote process via a uni- or bi-directional channel. Examples include communication with a SAT solver, featherweight Web (service) client, proxies and wrappers. MySysOpen supports Unix pipes, and Unix domain and TCP sockets. MySysOpen and the underlying `sys_open.c` have been used in production for many years, on Linux and various Unix platforms.

The included tests check sending and receiving of large amounts of data, and communicating with third-party programs such as a SAT solver via a bi-directional pipe. Generally, a program must be specifically written for interactive use over a bi-directional pipe: The program should avoid read-ahead, produce output as soon as it obtained all necessary input data, and be especially careful with buffering. Most systems programs are not written with these goals in mind. Our test uses Unix `sort`, which is particularly unsuitable for interaction: it cannot produce any output before it has read all of the input. It has no input terminator other than the EOF condition. Alas, to send EOF, we have to close the communication channel. Our test demonstrates two work-arounds, using shutdown(2) and a custom EOF indicator.

http://okmij.org/ftp/Haskell/misc.html#sys_open

1.5 Haskell Cheat Sheet

Report by:	Justin Bailey
Status:	active development

See: http://www.haskell.org/communities/05-2010/html/report.html#sect1.4.

1.6 A Tutorial on the Enumerator Library

Report by:	Kazu Yamamoto

Enumerator/Iteratee (EI) developed by Oleg Kiselyov is an API to enable modular programming in the IO monad. A popular implementation of EI is the *enumerator* library developed by John Millikin. This tutorial is a gentle introduction of the background of EI and how to use the *enumerator* library.

Further reading

http://www.mew.org/~kazu/proj/enumerator/

1.7 Practice of Functional Programming

Report by:	Dmitry Astapov
Status:	seven issues out, issue #8 is looming ahead, collecting materials for more

“Practice of Functional Programing” is a Russian electronic magazine promoting functional programming. The magazine features articles that cover both theoretical and practical aspects of the craft. Significant amount of the already published material is directly related to Haskell.

The magazine attempts to keep a bi-monthly release schedule, with Issue #7 leaving the press at the end of April 2011. Full contents of current and past issues are available in PDF from the official site of the magazine free of charge. Articles are in Russian, with English annotations.

Further reading

http://fprog.ru/ for issues ##1–7

2 Implementations

2.1 Haskell Platform

Report by: Duncan Coutts

Background

The Haskell Platform (HP) is the name of the "blessed" set of libraries and tools on which to build further Haskell libraries and applications. It takes a core selection of packages from the more than 3000 on Hackage (→ 5.8.1). It is intended to provide a comprehensive, stable, and quality tested base for Haskell projects to work from.

Historically, GHC shipped with a collection of packages under the name `extralibs`. Since GHC 6.12 the task of shipping an entire platform has been transferred to the Haskell Platform.

Recent progress

This spring we had the third major release of the platform. This is the 2011.1.0.x release series. This included the excellent new `text` package, a major upgrade to the `mtl` package and of course GHC 7.0.x.

Looking forward

Major releases take place on a 6 month cycle. The next major release will be in Autumn 2011 and will most likely include GHC 7.2.x.

This is was the first round where we went through the community review process to accept new packages into the platform. For one package this process went smoothly and for another it did not. The platform steering committee will be proposing some modifications to the process with the aim of reducing the burden for package authors and keeping the review discussions productive.

Though we will be making some modifications, we would still like to invite package authors to propose new packages. This can be initiated at any time. We also invite the rest of the community to take part in the review process on the libraries mailing list ⟨libraries@haskell.org⟩. The procedure involves writing a package proposal and discussing it on the mailing list with the aim of reaching a consensus. Details of the procedure are on the development wiki.

Further reading

http://haskell.org/haskellwiki/Haskell_Platform

- Download: http://hackage.haskell.org/platform/
- Wiki: http://trac.haskell.org/haskell-platform/
- Adding packages: http://trac.haskell.org/haskell-platform/wiki/AddingPackages

2.2 The Glasgow Haskell Compiler

Report by: Simon Peyton Jones
Participants: many others

GHC is still busy as ever. The GHC 7.0 branch has come and gone, and now that the branch has been closed we have finally made the long-planned switch from darcs to git. Meanwhile, we are busily working towards the 7.2 branch, and hope to make the 7.2.1 release in June. Some of the forthcoming highlights are:

- In the autumn, Dimitrios and Simon PJ implemented a completely new constraint solver for the type checker; we also complete an epic JFP paper describing how it works [OutsideIn]. The new solver is far more tractable and maintainable than the old type checker, and has fixed many outstanding problems. We are still shaking out the last bugs, and we have some nifty ideas for improving performance. Based on this new foundation, we are planning to develop the type system further, notably by adding a richer kind system along the lines of Conor McBride's SHE system [SHE].

- GHC's intermediate language (which we call "Core") is a simple, explicitly-typed lambda in the style of System F. Core is far, far simpler than Haskell (*CoreExpr* has only eight data constructors), so GHC can type-check Core very fast and reliably. In theory, such a typecheck is redundant (since the original Haskell program was typechecked), but in practice, typechecking Core is a very powerful internal consistency check on GHC itself: many compiler bugs generate type-incorrect Core. This consistency check is run by `-dcore-lint`.

  With the advent of GADTs and type families, the type system of the Core had to grow a little. For a few years we have been using an extension of System F, called System FC, as described in our 2007 paper [FC]. However, the way that System FC was actually *implemented* in GHC's Core language was a bit unsatisfactory so, with help from Brent Yorgey, Simon PJ is busy re-engineering it. In particular, FC has *coercion terms*, and these will now be represented by their own data type *Coercion*, rather than being squeezed into *Type*. Moreover, these coercion terms can get big, so there is a new "coercion optimiser" to replace big coercions by equivalent smaller ones. All this is described in our new paper [NewFC]. These changes will (finally) complete the type-family story by making so-called "equality superclasses" work for the first time in GHC 7.2.

- José Pedro Magalhães has nearly completed his implementation of the *derivable type classes* mechanism described in his 2010 Haskell Symposium paper [Derivable] and elsewhere in this report (→ 5.5.1). It will be in GHC 7.2.

- Edward Yang has spearheaded a flurry of work on the new code generation backend (`-fuse-new-codegen`, the rewrite of the part of GHC that turns STG syntax into C–). Hoopl is now fully part of GHC [Hoopl], and the new path uses it extensively; we have ironed out most of the bugs in the backend; and now we are working on new optimization passes and fixing inefficiencies to get the generated code as good (or better) than the old code generator. We are still not at the point where the new code generator will generate better code, but we are pretty close! Stay tuned.

- Simon Marlow and Ryan Newton have developed a neat new library for deterministic parallel programming in Haskell; read their ICFP submission [DetPar]. The model is monadic and has explicit control over granularity, but allows dynamic construction of dataflow networks that are scheduled at runtime, while remaining deterministic and pure.

- Simon Marlow has been busy implementing and benchmarking a new garbage collector. GHC's current garbage collector is of the parallel "stop-the-world" variety, where to collect the heap all cores stop running the program and collect the heap in parallel. The new collector is a "local heap" collector, in which each core has a private heap that can be collected independently of the other cores, meanwhile there is a shared global heap that is collected (much less frequently) by the usual parallel stop-the-world algorithm. We have a paper describing the new design which has been accepted at ISMM'11 (and will be online shortly). The results are mixed; while on average performance improves with the new collector for parallel programs, the improvements are not dramatic (at least up to 24 cores). The new collector is significantly more complex than GHC's current collector. Hence we do not plan to merge it into the mainline yet, but will maintain it on a git branch for the time being, while we continue to experiment with and tune it. Some improvements from the branch that were independent of the new GC algorithm have already been merged into the mainline, so 7.2.1 will see some small improvements in GC performance and stats reporting.

- Simon Marlow has implemented a chunked stack representation, which should improve the performance of programs that need large stacks. See the [ChunkedStack]. This is already in the mainline and will be in the 7.2.1 release.

We are fortunate to have a growing team of people willing to roll up their sleeves and help us with GHC. Amongst those who have been active recently are:
- Mark Lentczner and Dan Knapp have been working on cross-compilation support
- Continued work on the new I/O manager by Johan Tibell
- Various improvements and build fixes for OS X, from PHO, Greg Wright, Thorkil Naur and William Knop
- Solaris fixes from Karel Gardas and Christian Maeder
- Gentoo fixes (for SE Linux and x86 FreeBSD support) from Sergei Trofimovich
- Other FreeBSD fixes from Marco Silva
- Linux PowerPC fixes from Erik de Castro Lopo
- Objective C support has been added by Austin Seipp
- Documentation updates from Orphi
- Various improvements from Michal Terepeta
- General tidyups from Matthias Kilian
- Primop improvements from Daniel Peebles
- Some GHCi improvements from Vivian McPhail and Boris Lykah
- More GHCi debugger fixes from Pepe Iborra
- LLVM development continues with David Terei
- Many people have given git help to those of us new to git

At GHC HQ we are having way too much fun; if you wait for us to do something you have to wait a long time. So do not wait; join in!

Other developments

GHC continues to act as an incubator for interesting new language developments. Here is a selection that we know about.

- Jeff Epstein, in collaboration with Andrew Black, has implemented a library that brings Erlang's programming model to Haskell programmers. In particular, you can write a Haskell program that runs on a cluster of machines that do not share memory. It is all based on a modest but powerful language extension that makes it possible for a programmer to work with "static" functions; that is, ones consisting of pure code with no free variables. The paper that describes all this is called "Haskell for the cloud" [Cloud].

- Max Bolingbroke continues his PhD work on supercompilation, with a nice new paper [ImprovingSupercompilation]. The plan is to make his supercompiler part of GHC, over the next year or so.

- David Terei at Stanford is busy implementing "Safe Haskell", a flag for GHC that will guarantee that your program has certain properties such as referential transparency and constructor access control, while still having the same semantics as it normally would. The flag basically allows you to trust the types of your program, giving you if you will a more "pure" version of Haskell where *unsafePerformIO* is outlawed, abstract data types are actually abstract and safety is provided by the compiler not the user. This is being done as part of a larger project by the Stanford Secure Computing Systems group involving the use of dynamic information flow based security in Haskell to build a secure web framework that allows the inclusion of third party untrusted code.

- Ranjit Jhala at UC San Diego is working on implementing Liquid Types [Liquid] within GHC. The goal is to allow programmers to use lightweight refinement types to specify key invariants which can then be verified through a combination of type inference and SMT solving.

The Parallel GHC Project

Microsoft Research is funding a 2-year project to develop the real-world use of parallel Haskell. The project is now underway with four industrial partners:
- Dragonfly (New Zealand)
- IIJ Innovation Institute Inc. (Japan)
- Los Alamos National Laboratory (USA)
- Willow Garage Inc. (USA)

with consulting and engineering support from Well-Typed (→8.1). Each organisation is working on its own particular project making use of parallel Haskell. The overall goal is to demonstrate successful serious use of parallel Haskell, and along the way to apply engineering effort to any problems with the tools that the organisations might run into.

For more details, see the Parallel GHC Project entry (→4.1.5), and the project home page [ParallelGhcProject]

Data Parallel Haskell

The main user-visible development concerning data-parallel programming with GHC since the last status report is the release of our library for regular, multi-dimensional, shape-polymorphic arrays: [Repa]. The current release on Hackage performs well with GHC 7.0.3 and already includes Ben's recent work on high-performance stencil-based convolutions — see also the draft paper [Stencil] and Ben's screencast [EdgeDetect] of a real-time edge detection application, written in Objective-C and Haskell, using the new Repa library.

We have pushed back the release of a stable version of the main DPH libraries again. They are now scheduled to be released with the forthcoming GHC 7.2.

Bibliography

ChunkedStack "An overhaul of stack management, and some performance improvements", Simon Marlow, blog post, Dec2010, http://hackage.haskell.org/trac/ghc/blog/stack-chunks

Cloud "Haskell for the cloud", Epstein, Black, Peyton Jones, submitted to ICFP 2011, http://research.microsoft.com/~simonpj/papers/parallel/

Derivable "A generic deriving mechanism for Haskell", Magalhães, Dijkstra, Jeuring and Löh, Haskell Symposium 2010, http://www.dreixel.net/research/pdf/gdmh_nocolor.pdf

DetPar "A monad for deterministic parallelism", Marlow, Newton, and Peyton Jones, submitted to ICFP 2011, http://research.microsoft.com/~simonpj/papers/parallel/

EdgeDetect "Edge-detection video", http://code.ouroborus.net/beholder/video/Edge480.mov

FC "System F with type equality coercions", Sulzmann, Chakravarty, Peyton Jones, TLDI 2007, http://research.microsoft.com/~simonpj/papers/ext-f/

Hoopl "A modular, reusable library for dataflow analysis and transformation", Dias, Ramsey, and Peyton Jones, Haskell Symposium 2010, http://research.microsoft.com/~simonpj/papers/c--/

ImprovingSupercompilation "Improving supercompilation: tag-bags, rollback, speculation, normalisation, and generalisation", Bolingbroke and Peyton Jones, submitted to ICFP 2011, http://research.microsoft.com/~simonpj/papers/supercompilation/

Liquid "Liquid types", Ranjit Jhala, http://goto.ucsd.edu/~rjhala/liquid

NewFC "Practical aspects of evidence-based compilation in System FC", Vytiniotis and Peyton Jones, submitted to ICFP 2011, http://research.microsoft.com/~simonpj/papers/ext-f/

OutsideIn "Modular type inference with local assumptions", Vytiniotis, Peyton Jones, Schrijvers, and Sulzmann, Journal of Functional Programming (to appear), http://research.microsoft.com/~simonpj/papers/constraints/

ParallelGhcProject "The Parallel GHC Project home page", http://www.haskell.org/haskellwiki/Parallel_GHC_Project

Repa "Regular, shape-polymorphic parallel arrays in Haskell", Keller, Chakravarty, Leshchinskiy, Peyton Jones, and Lippmeier, ICFP 2010. Paper: http://research.microsoft.com/~simonpj/papers/ndp/, Hackage package: http://hackage.haskell.org/package/repa

SHE "The Strathclyde Haskell Enhancement", Conor McBride, 2010, http://personal.cis.strath.ac.uk/~conor/pub/she/

Stencil "Efficient Parallel Stencil Convolution in Haskell", Lippmeier et al., http://www.cse.unsw.edu.au/~benl/papers/stencil/stencil-icfp2011-sub.pdf

2.3 Immix Garbage Collector on GHC

Report by:	Marco Gontijo
Status:	unconcluded

During the summer of 2010, Marco Silva e Gontijo worked on the implementation of the Immix algorithm in GHC. Immix is a relatively new technique for garbage collection, which has been shown to be better than other alternatives, including the ones used in GHC. The work was done as a project in the Google Summer of Code.

The code is functional and does not contain known bugs. It gets better results than the default GC in the nofib suite. On the other hand, it gets worse results than the default GC for the nofib/gc suite. This scenario may change if more tuning is done in the details of the implementation. Given that GHC allows the user to choose between garbage collection alternatives at runtime, it is easy to test and compare the different techniques.

Immix was implemented using the experimental code from mark-sweep as a base. Currently, it overrides mark-sweep so that it is not that easy to compare immix with mark-sweep. The plan is to split them apart in the future.

On the GHC Commentary there is a page about the current state, with a to do list. There are some fundamental parts that are not yet implemented and that may improve performance, such as the allocation in lines in minor GCs and the removal of partial lists, which are not necessary in Immix.

Further reading

http://hackage.haskell.org/trac/ghc/wiki/Commentary/Rts/Storage/GC/Immix

2.4 UHC, Utrecht Haskell Compiler

Report by:	Atze Dijkstra
Participants:	many others
Status:	active development

What is new? UHC is the Utrecht Haskell Compiler, supporting almost all Haskell98 features and most of Haskell2010, plus experimental extensions. Since the last release a Javascript backend has been implemented. We plan to make a next release autumn this year.

What do we currently do and/or has recently been completed? As part of the UHC project, the following (student) projects and other activities are underway (in arbitrary order):

- Jeroen Bransen (PhD): "Incremental Global Analysis".
- Jan Rochel (PhD): "Realising Optimal Sharing", based on work by Vincent van Oostrum and Clemens Grabmayer.
- Arie Middelkoop (PhD, to be defended soon): type system formalization and automatic generation from type rules, in particular the Attribute Grammar variants Ruler-Core for supporting more complex type system implementations.
- Tamar Christina: an implementation of HML using Ruler-Core.
- Jeroen Leeuwestein: incrementalization of whole program analysis.
- Jeroen Fokker: GRIN backend, whole program analysis.
- Doaitse Swierstra: parser combinator library.
- Atze Dijkstra: overall architecture, type system, bytecode interpreter + java + javascript backend, garbage collector.

Background UHC actually is a series of compilers of which the last is UHC, plus infrastructure for facilitating experimentation and extension. The distinguishing features for dealing with the complexity of the compiler and for experimentation are (1) its stepwise organisation as a series of increasingly more complex standalone compilers, the use of DSL and tools for its (2) aspectwise organisation (called Shuffle) and (3) tree-oriented programming (Attribute Grammars, by way of the Utrecht University Attribute Grammar (UUAG) system (→ 4.4.1).

Further reading

- UHC Homepage: http://www.cs.uu.nl/wiki/UHC/WebHome
- UHC Blog: http://utrechthaskellcompiler.wordpress.com
- 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

2.5 Exchanging Sources between Clean and Haskell

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

In a Haskell'10 paper we describe how we facilitate the exchange of sources between Clean (→3.3) and Haskell. We use the existing Clean compiler as starting point, and implement a double-edged front end for this compiler: it supports both standard Clean 2.1 and (currently a large part of) standard Haskell 98. Moreover, it allows both languages to seamlessly use many of each other's language features that were alien to each other before. For instance, Haskell can now use uniqueness typing anywhere, and Clean can use newtypes efficiently. This has given birth to two new dialects of Clean and Haskell, dubbed Clean∗ and Haskell∗. Measurements of the performance of the new compiler indicate that it is on par with the flagship Haskell compiler GHC.

Future plans

Although the most important features of Haskell 98 have been implemented, the list of remaining issues is still rather long since some features took much more work than expected. Also, to enable the practical reuse of Haskell libraries, we have to implement some of GHC's extensions, such as generalised algebraic datatypes and type families. This is challenging, not only in terms of the programming effort, but more because of the consequences it will have on features such as uniqueness typing. We plan to use this double-edged front as an implementation laboratory to investigate these avenues.

Further reading

- John van Groningen, Thomas van Noort, Peter Achten, Pieter Koopman, and Rinus Plasmeijer. Exchanging sources between Clean and Haskell — A double-edged front end for the Clean compiler. In Jeremy Gibbons, editor, *Proceedings of the Haskell Symposium, Haskell '10, Baltimore, MD, USA*, pages 49–60. ACM Press, 2010.
- The front end is under active development, current releases are available via http://wiki.clean.cs.ru.nl/Download_Clean.

2.6 The Reduceron

Report by:	Matthew Naylor
Participants:	Colin Runciman, Jason Reich, Marco Perez Cervantes
Status:	experimental

The Reduceron is a graph-reduction processor implemented on an FPGA.

Between May 2009 and November 2010, work on the Reduceron has led to a factor of five speed-up. This has been achieved through a range of design improvements spanning architectural, machine, and compiler-level issues. See our ICFP'10 paper for details.

Work on the Reduceron continues. We have taken a step towards parallel reduction in the form of *primitive redex speculation*. We have developed a static analysis and transformation (currently limited to first-order programs) that predicts and increases run-time occurrences of primitive redexes, allowing a simpler and faster machine design. Early results look good, and we hope to extend the technique to higher-order programs.

Experiments in verification, both at the compiler level and the bytecode level, are also underway.

Looking ahead, we aim eventually to have multiple Reducerons running in parallel. We are also interested in increasing the amount of memory available to the Reduceron, and in technology advances that may enable faster clocking frequencies.

Two main by-products have emerged from the work. First, *York Lava*, now available from Hackage, is the HDL we use. It is very similar to Chalmers Lava (→9.9), but supports a greater variety of primitive components, behavioral description, number-parameterized types, and a first attempt at a Lava prelude. Second, *F-lite* is our subset of Haskell, with its own lightweight toolset and experimental supercompiler (http://haskell.org/communities/11-2009/html/report.html#sect4.1.4).

Further reading

- http://www.cs.york.ac.uk/fp/reduceron/
- http://hackage.haskell.org/package/york-lava/

2.7 Specific Platforms

2.7.1 Haskell on FreeBSD

Report by:	PÁLI Gábor János
Participants:	FreeBSD Haskell Team
Status:	ongoing

The FreeBSD Haskell Team is a small group of contributors who maintain Haskell software on all actively supported versions of FreeBSD. The primarily supported implementation is the Glasgow Haskell Compiler together with Haskell Cabal, although one may also find Hugs and NHC98 in the Ports Collection.

FreeBSD has become a Tier-1 platform for GHC in April 2010 (on both i386 and amd64), and starting from GHC 6.12.1, one can download vanilla binary distributions for each release. In addition, we have an experimental project, called “hsporter” to help conversion of existing Cabal packages to FreeBSD ports.

We also created a developer repository for Haskell ports that now includes around 200 ported packages, featuring the latest version of many popular Cabal packages. The updates committed to this repository are continuously integrated to the Ports Collection as they become stable. We expect smoother and more regular updates in the future.

We have recently merged most of our new and updated ports back to the official tree, so it now has GHC 7.0.3, Haskell Platform 2011.2.0.1, Gtk2Hs 0.12, XMonad 0.9.2, Pandoc 1.8, and Darcs 2.5.

If you find yourself interested in helping us or simply want to use the latest versions of Haskell programs on FreeBSD, check out our page at the FreeBSD wiki (see below) where you can find all important pointers and information required for use, contact, or contribution.

Further reading

http://wiki.FreeBSD.org/Haskell

2.7.2 Debian Haskell Group

Report by:	Joachim Breitner
Status:	working

The Debian Haskell Group aims to provide an optimal Haskell experience to users of the Debian GNU/Linux distribution and derived distributions such as Ubuntu. We try to follow the Haskell Platform versions for the core package and package a wide range of other useful libraries and programs. In total, we maintain 215 source packages.

A system of virtual package names and dependencies, based on the ABI hashes, guarantees that a system upgrade will leave all installed libraries usable. Most libraries are also optionally available with the profiling data and the documentation packages register with the system-wide index.

While writing these lines, we are in the progress of transitioning to ghc version 7, of which the Haskell Group has become the maintainer as well. While doing that, we drop the 6 from the library package names, which causes the transition to take longer than usual. Nevertheless, the Haskell Platform is available in Debian unstable in version 2011.2.0.1, while the recently released stable version of Debian, “squeeze”, ships 2010.1.0.0.

Further reading

http://wiki.debian.org/Haskell

2.7.3 Haskell in Gentoo Linux

Report by:	Sergei Trofimovich

Gentoo Linux currently officially supports GHC 6.12.3 on x86, amd64, sparc, ppc, ppc64, alpha and ia64. Hppa support was dropped.

GHC also runs on gentoo-hardened http://www.gentoo.org/proj/en/hardened/ and on some gentoo-alt http://www.gentoo.org/proj/en/gentoo-alt/ systems. They are freebsd, macos-prefix and solaris-prefix for now. Special thanks to Fabian Groffen and the Prefix Team.

The full list of packages available through the official repository can be viewed 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 an overlay which contains more than 600 extra unofficial and testing packages. Thanks to the Haskell developers using Cabal and Hackage (→ 5.8.1), we have been able to write a tool called “hackport” (initiated by Henning Günther) to generate Gentoo packages with minimal user intervention. Notable packages in the overlay include the latest version of the Haskell Platform (→ 2.1) as well as the latest 7.0.3 release of GHC, as well as popular Haskell packages such as pandoc (→ 7.2.3) and gitit (→ 4.2.5).

Due to tremendous amount of work done by Mark Wright most of the packages work with GHC 7.0.3.

All Gentoo Haskell projects moved to https://github.com/gentoo-haskell where one can find the new home of our overlay and tools helping to keep the overlay up-to-date.

More information about the Gentoo Haskell Overlay can be found at http://haskell.org/haskellwiki/Gentoo. It is available via the Gentoo overlay manager “layman”. If you choose to use the overlay, then any problems should be reported on IRC (`#gentoo-haskell` on freenode), where we coordinate development, or via email ⟨haskell@gentoo.org⟩ (as we have more people with the ability to fix the overlay packages that are contactable in the IRC channel than via the bug tracker).

As always we are more than happy for (and in fact encourage) Gentoo users to get involved and help us maintain our tools and packages, even if it is as simple as reporting packages that do not always work or need updating: with such a wide range of GHC and package versions to co-ordinate, it is hard to keep up! Please contact us on IRC or email if you are interested!

2.7.4 Fedora Haskell SIG

Report by:	Jens Petersen
Participants:	Lakshmi Narasimhan, Ben Boeckel, Shakthi Kannan, Bryan O'Sullivan, and others
Status:	ongoing

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

Fedora 15 is scheduled to ship at the end of May with ghc-7.0.2, haskell-platform-2011.2.0.0, and darcs-2.5.2. There are some major packaging improvements:

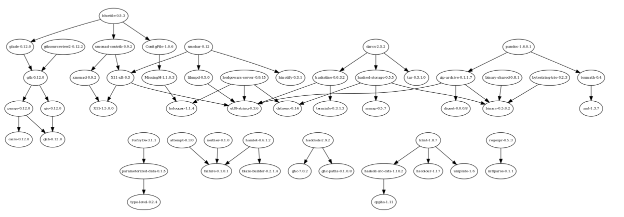
- All libraries from GHC are now subpackaged: this is good for Fedora which ships shared Haskell libraries.
- GHC package hash metadata has been added to all the binary packages for Fedora 15 to ensure consistency of library dependencies at build- and run-time.

The Fedora Haskell Packaging Guidelines are being updated and revised: a draft is currently under review.

Newly added packages this time include pandoc, bluetile, and over 35 new libraries.

There are currently 106 Haskell-related source packages in Fedora, and about 30 new packages in the review queue. Our packages are now also listed on the Hackage website.

Here is a graph of the package dependencies in Fedora 15 (with ghc and haskell-platform packages factored out):



In the Fedora 16 cycle we may update to ghc-7.0.3 and will add more packages: including leksah and hopefully a web framework.

Contributions to Fedora Haskell are welcome: join us on #fedora-haskell on Freenode IRC and our mailing-list.

Further reading

- Homepage: http://fedoraproject.org/wiki/SIGs/Haskell
- Fedora 15 Haskell release-notes: http://fedoraproject.org/wiki/Documentation_Development_Haskell_Beat
- Package list: https://admin.fedoraproject.org/pkgdb/users/packages/haskell-sig?tg_paginate_limit=0
- Open package reviews: https://bugzilla.redhat.com/showdependencytree.cgi?id=Haskell-pkg-reviews&hide_resolved=1
- Revision of Packaging Guidelines: https://fedoraproject.org/wiki/PackagingDrafts/Haskell
- Dependency graphs: https://fedoraproject.org/wiki/Haskell_package_interdependencies

2.8 Fibon Benchmark Tools & Suite

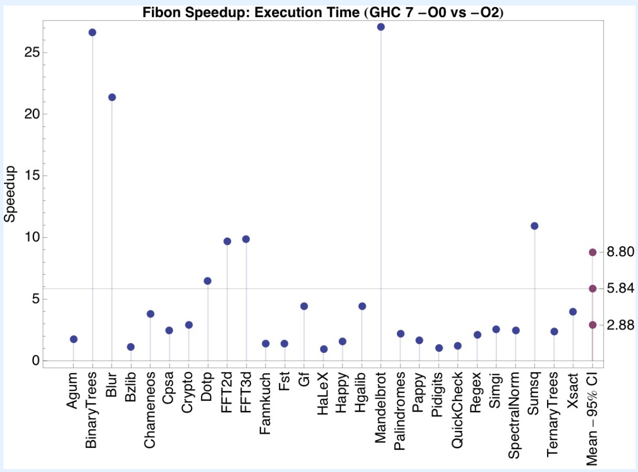
Report by:	David M. Peixotto
Status:	stable

Fibon is a set of tools for running and analyzing benchmark programs in Haskell. It contains an optional set of benchmarks from various sources including several programs from the Hackage repository.

The Fibon benchmark tools draw inspiration from both the venerable nofib Haskell benchmark suite and the industry standard SPEC benchmark suite. The tools automate the tedious parts of benchmarking: building the benchmark in a sand-boxed directory, running the benchmark multiple times, verifying correctness, collecting statistics, and summarizing results.

Benchmarks are built using the standard `cabal` tool. Any program that has been cabalized can be added as benchmark simply by specifying some meta-information about the program inputs and expected outputs. Fibon will automatically collect execution times for benchmarks and can optionally read the statistics output by the GHC runtime. The program outputs are checked to ensure correct results making Fibon a good option for testing the safety and performance of program optimizations. The Fibon tools are not tied to any one benchmark suite. As long as the correct meta-information has been supplied, the tools will work with any set of programs.

As a real life example of a complete benchmark suite, Fibon comes with its own set of benchmarks for testing the effectiveness of compiler optimizations in GHC. The benchmark programs come from Hackage, the Computer Language Shootout, Data Parallel Haskell, and Repa. The benchmarks were selected to have minimal external dependencies so they could be easily used with a version of GHC compiled from the latest sources. The following figure shows the performance improvement of GHC's optimizations on the Fibon benchmark suite.



The Fibon tools and benchmark suite are ready for public consumption. They can be found on github at the url indicated below. People are invited to use the included benchmark suite or just use the tools and build a suite of their own creation. Any improvements to the tools or additional benchmarks are most welcome. Benchmarks have been used to tell lies for many years, so join in the fun and keep on fibbing with Fibon.

Further reading

- https://github.com/dmpots/fibon
- https://github.com/dmpots/fibon-benchmarks
- https://github.com/dmpots/fibon-config

3 Related Languages

3.1 Agda

Report by:	Nils Anders Danielsson
Participants:	Ulf Norell, Andreas Abel, and many others
Status:	actively developed

Agda is a dependently typed functional programming language (developed using Haskell). A central feature of Agda is inductive families, i.e. GADTs which can be indexed by *values* and not just types. The language also supports coinductive types, parameterized modules, and mixfix operators, and comes with an *interactive* 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.

In February version 2.2.10 was released. This release includes a new compiler backend, implemented by Daniel Gustafsson and Olle Fredriksson. The backend incorporates several new optimisations, based on work by Edwin Brady and others, and work is in progress to add even more optimisations.

Further reading

The Agda Wiki: http://wiki.portal.chalmers.se/agda/

3.2 MiniAgda

Report by:	Andreas Abel
Status:	experimental

MiniAgda is a tiny dependently-typed programming language in the style of Agda (→ 3.1). It serves as a laboratory to test potential additions to the language and type system of Agda. MiniAgda's termination checker is a fusion of sized types and size-change termination and supports coinduction. Equality incorporates eta-expansion at record and singleton types. Function arguments can be declared as static; such arguments are discarded during equality checking and compilation.

Recent features include bounded size quantification and destructor patterns for a more general handling of coinduction. In the long run, I plan to evolve MiniAgda into a core language for Agda with termination certificates.

MiniAgda is available as Haskell source code and compiles with GHC $\geqslant$ 6.12.x.

Further reading

http://www2.tcs.ifi.lmu.de/~abel/miniagda/

3.3 Clean

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

Clean is a general purpose, state-of-the-art, pure and lazy functional programming language designed for making real-world applications. Here is a short list of notable features:

- Clean is a lazy, pure, and 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, type synonyms, and existentially quantified types.
- Clean has pattern matching, guards, list comprehensions, array comprehensions and a lay-out sensitive mode.
- Clean supports type classes and type constructor classes to make overloaded use of functions and operators possible.
- The uniqueness typing system in Clean makes it possible to develop efficient applications. In particular, it allows a refined control over the single-threaded use of objects which can influence the time and space behavior of programs. Uniqueness typing can also be 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 nonfunctional world (to C but also to I/O systems like X-Windows) offering direct access to file systems and operating systems.
- Clean offers records and (destructively updateable) arrays and files.

- The Clean type system supports dynamic typing, allowing values of arbitrary types to be wrapped in a uniform package and unwrapped via a type annotation at run time. Using dynamics, code and data can be exchanged between Clean applications in a flexible and type-safe way.
- Clean provides a built-in mechanism for generic functions.
- There is a Clean IDE and there are many libraries available offering additional functionality.
- There is (experimental) support for the exchange of sources between Clean and Haskell, please see the corresponding entry (→ 2.5) for more information.

Future plans

- We are currently working on the generic function mechanism: we are improving efficiency and including support for generic dependencies, the latter allows us to use arbitrary generic functions on the type parameters of a generic type argument.
- Clean is already available for 32-bit and 64-bit Windows and Linux, we are currently working on 64-bit Mac support.
- Please see the entry on exchanging sources between Clean and Haskell (→ 2.5) for more future plans.

Further reading

- http://wiki.clean.cs.ru.nl/
- http://wiki.clean.cs.ru.nl/Download_Clean

3.4 Timber

Report by:	Johan Nordlander
Participants:	Björn von Sydow, Andy Gill, Magnus Carlsson, Per Lindgren, Thomas Hallgren, and others
Status:	actively developed

Timber is a general programming language derived from Haskell, with the specific aim of supporting development of complex event-driven systems. It allows programs to be conveniently structured in terms of objects and reactions, and the real-time behavior of reactions can furthermore be precisely controlled via platform-independent timing constraints. This property makes Timber particularly suited to both the specification and the implementation of real-time embedded systems. An implementation of Timber is available as a command-line compiler tool, currently targeting POSIX-based systems only.

Timber shares most of Haskell's syntax but introduces new primitive constructs for defining classes of reactive objects and their methods. These constructs live in the *Cmd* monad, which is a replacement of Haskell's top-level monad offering mutable encapsulated state, implicit concurrency with automatic mutual exclusion, synchronous as well as asynchronous communication, and deadline-based scheduling. In addition, the Timber type system supports nominal subtyping between records as well as datatypes, in the style of its precursor O'Haskell.

A particularly notable difference between Haskell and Timber is that Timber uses a *strict* evaluation order. This choice has primarily been motivated by a desire to facilitate more predictable execution times, but it also brings Timber closer to the efficiency of traditional execution models. Still, Timber retains the purely functional characteristic of Haskell, and also supports construction of recursive structures of arbitrary type in a declarative way.

The Timber compiler is currently undergoing a major reimplementation of its front-end, an effort triggered by increasing needs to significantly improve error messages as well as to sharpen up the documentation of the language syntax and its scoping rules. Regrettably, no visible developments of this undertaking can be reported since the November 2010 issue of HCAR. Work on the new compiler continues, however, with the aim of releasing a version 2 before the end of 2011. The current release of the Timber compiler system dates back to May 2009 (version 1.0.3).

Further reading

http:://timber-lang.org

3.5 Disciple

Report by:	Ben Lippmeier
Participants:	Erik de Castro Lopo
Status:	experimental, active development

Disciple is a dialect of Haskell that uses strict evaluation as the default and supports destructive update of arbitrary data. Many Haskell programs are also Disciple programs, or will run with minor changes. In addition, Disciple includes region, effect, and closure typing, and this extra information provides a handle on the operational behaviour of code that is not available in other languages. Our target applications are the ones that you always find yourself writing C programs for, because existing functional languages are too slow, use too much memory, or do not let you update the data that you need to.

Our compiler (DDC) is still in the "research prototype" stage, meaning that it will compile programs if you are nice to it, but expect compiler panics and missing features. You will get panics due to ungraceful handling of errors in the source code, but valid programs should compile ok. The test suite includes a few thousand-line graphical demos, like a ray-tracer and an n-body collision simulation, so it is definitely hackable.

Over the last six months Erik has continued work on the LLVM backend, which is almost finished now. It compiles all the programs in the test-suite, with just a few hacky things in the DDC base library needing to be fixed. In the meantime, I have started to mechanise the proofs of the core language in Coq, as the old latex proofs were just getting too big to manage by hand. I have made it through Progress and Preservation for System-F just using vanilla de Bruijn indices for binders, and am starting to add the features specific to DDC core now.

Further reading

http://disciple.ouroborus.net

4 Haskell and . . .

4.1 Haskell and Parallelism

4.1.1 TwilightSTM

Report by:	Annette Bieniusa
Participants:	Arie Middelkoop, Peter Thiemann
Status:	experimental

TwilightSTM is an extended Software Transactional Memory system. It safely augments the STM monad with non-reversible actions and allows introspection and modification of a transaction's state.

TwilightSTM splits the code of a transaction into a (functional) atomic phase, which behaves as in GHC's implementation, and an (imperative) twilight phase. Code in the twilight phase executes before the decision about a transaction's fate (restart or commit) is made and can affect its outcome based on the actual state of the execution environment.

The Twilight API has operations to detect and repair read inconsistencies as well as operations to overwrite previously written variables. It also permits the safe embedding of I/O operations with the guarantee that each I/O operation is executed only once. In contrast to other implementations of irrevocable transactions, twilight code may run concurrently with other transactions including their twilight code in a safe way. However, the programmer is obliged to prevent deadlocks and race conditions when integrating I/O operations that participate in locking schemes.

A prototype implementation is available on Hackage (http://hackage.haskell.org/package/twilight-stm). We are currently working on the composability of Twilight monads and are applying TwilightSTM to different use cases.

Further reading

http://proglang.informatik.uni-freiburg.de/projects/twilight/

4.1.2 Haskell-MPI

Report by:	Bernie Pope
Participants:	Dmitry Astapov, Duncan Coutts
Status:	first public version to be released soon

MPI, the *Message Passing Interface*, is a popular communications protocol for distributed parallel computing (http://www.mpi-forum.org/). It is widely used in high performance scientific computing, and is designed to scale up from small multi-core personal computers to massively parallel supercomputers. MPI applications consist of independent computing processes which share information by message passing communication. It supports both point-to-point and collective communication operators, and manages much of the mundane aspects of message delivery. There are several high-quality implementations of MPI available which adhere to the standard API specification (the latest version of which is 2.2). The MPI specification defines interfaces for C, C++, and Fortran, and bindings are available for many other programming languages. As the name suggests, Haskell-MPI provides a Haskell interface to MPI, and thus facilitates distributed parallel programming in Haskell. It is implemented on top of the C API via Haskell's foreign function interface. Haskell-MPI provides three different ways to access MPI's functionality:

1. A direct binding to the C interface.
2. A convenient interface for sending arbitrary serializable Haskell data values as messages.
3. A high-performance interface for working with (possibly mutable) arrays of storable Haskell data types.

We do not currently provide exhaustive coverage of all the functions and types defined by MPI 2.2, although we do provide bindings to the most commonly used parts. In the future we plan to extend coverage based on the needs of projects which use the library.

We are in the final stages of preparing the first release of Haskell-MPI. We will publish the code on Hackage once the user documentation is complete. We have run various simple latency and bandwidth tests using up to 512 Intel x86-64 cores, and for the high-performance interface, the results are within acceptable bounds of those achieved by C. Haskell-MPI is designed to work with any compliant implementation of MPI, and we have successfully tested it with both OpenMPI (http://www.open-mpi.org/) and MPICH2 (http://www.mcs.anl.gov/research/projects/mpich2/).

Further reading

http://github.com/bjpop/haskell-mpi

4.1.3 Eden

Report by:	Rita Loogen
Participants:	**in Madrid:** Yolanda Ortega-Mallén, Mercedes Hidalgo, Lidia Sánchez-Gil, Fernando Rubio, Alberto de la Encina, **in Marburg:** Mischa Dieterle, Thomas Horstmeyer, Dominik Krappel, Oleg Lobachev, Rita Loogen, Bernhard Pickenbrock, Tobias Sauerwein **in Copenhagen:** Jost Berthold
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 new Eden logo

consists of four λ turned in such a way that they form the Eden instantiation operator #. Higher-level coordination is achieved by defining *skeletons*, ranging from a simple parallel map to sophisticated master-worker schemes. They have been used to parallelize a set of non-trivial programs.

Recently we have extended Eden's interface to support a simple definition of arbitrary communication topologies using *Remote Data*. Also, a new *PA*-monad enables the *eager* execution of user defined sequences of *Parallel Actions* in Eden.

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

The current release of the Eden compiler based on GHC 6.12.3 is available on our web pages, see http://www.mathematik.uni-marburg.de/~eden.

The next update will include a shared memory mode which does not depend on a middleware like MPI but which nevertheless uses multiple independent heaps (in contrast to GHCs threaded runtime system) connected by Eden's parallel runtime system. A compiler version based on GHC 7.0.3 is in its final testing phase.

Tools and libraries

The Eden trace viewer tool EdenTV was further developed to enhance its performance, usability and compatibility through newer eventlog format versions. This tool has been written in Haskell and is also freely available on the Eden web pages.

The Eden skeleton library is under constant development. It is available on the Eden pages.

Recent and Forthcoming Publications

- J. Berthold: *Orthogonal Serialisation for Haskell*, 22nd Symposium on Implementation and Application of Functional Languages (IFL 2010), Springer LNCS (to appear), 2011.
- C. Brown, H.-W. Loidl, J. Berthold, and K. Hammond: *Improve your CASH flow: The Computer Algebra SHell*, In 22nd Symposium on Implementation and Application of Functional Languages (IFL 2010), Springer LNCS (to appear), 2011.
- R. Loogen: *Eden*, Entry for the Springer Encyclopedia of Parallel Computing, Springer 2011, to appear.
- B. Pickenbrock: *A Multicore Implementation of Eden*, Bachelor Thesis, Philipps-Universität Marburg, 2011 (in German).
- L. Sánchez-Gil, M. Hidalgo-Herrero, and Y. Ortega-Mallén: *Relating function spaces to resourced function spaces*, Proceedings of the 26th Symposium on Applied Computing 2011 (SAC 2011), ACM 2011, 1301–1308.

Further reading

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

4.1.4 GpH — Glasgow Parallel Haskell

Report by:	Hans-Wolfgang Loidl
Participants:	Phil Trinder, Patrick Maier, Mustafa Aswad, Malak Aljabri, Robert Stewart (Heriot-Watt University); Kevin Hammond, Vladimir Janjic, Chris Brown (St Andrews University)
Status:	ongoing

Status

A distributed-memory, GHC-based implementation of the parallel Haskell extension GpH and of a fundamentally revised version of the evaluation strategies abstraction is available in a prototype version. In current research an extended set of primitives, supporting hierarchical architectures of parallel machines, and extensions of the runtime-system for supporting these architectures are being developed.

System Evaluation and Enhancement

- Both GpH and Eden (→ 4.1.3) parallel Haskells are being used for parallel language research and in the SCIEnce and HPC-GAP projects (see below).

- We are extending the set of primitives for parallelism to better control data locality.

- We are revising the evaluation strategies abstraction for improved genericity.

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

GpH Applications

As part of the SCIEnce EU FP6 I3 project (026133) (April 2006 – December 2011) and the HPC-GAP project (October 2009 – September 2013) we use Eden and GpH as middleware to provide access to computational Grids from Computer Algebra (CA) systems, including GAP, Maple MuPad and KANT. We have developed and released SymGrid-Par, a Haskell-side infrastructure for orchestrating heterogeneous computations across high-performance computational Grids. Based on this infrastructure we have developed a range of domain-specific parallel skeletons for parallelising representative symbolic computation applications. We are currently extending SymGrid-Par with support for fault-tolerance, targeting massively parallel high-performance architectures.

In recent work we have developed and released a GHCi-based computer algebra shell (CASH) that gives direct access to computer algebra functionality, provided by an SCSCP server, and enabling easy parallelism on the Haskell side.

Implementations

The latest GUM implementation of GpH is built on GHC 6.12, using either PVM or MPI as communications library. It implements a virtual shared memory abstraction over a collection of physically distributed machines. At the moment our main hardware platforms are Intel-based Beowulf clusters of multicores. We plan to connect several of these clusters into a wide-area, hierarchical, heterogenous parallel architecture.

Further reading

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

Contact

⟨gph@macs.hw.ac.uk⟩

4.1.5 Parallel GHC Project

Report by:	Eric Kow
Participants:	Duncan Coutts, Andres Löh, Spencer Janssen
Status:	active

Microsoft Research is funding a 2-year project to promote the real-world use of parallel Haskell. The project started in November 2010, with four industrial partners, and consulting and engineering support from Well-Typed (→ 8.1). Each organisation is working on its own particular project making use of parallel Haskell. The overall goal is to demonstrate successful serious use of parallel Haskell, and along the way to apply engineering effort to any problems with the tools that the organisations might run into.

The participating organisations are working on a diverse set of complex real world problems:

- Dragonfly (New Zealand): Hierarchichal Bayesian Modeling
- Los Alamos National Laboratory (USA): high performance Monte Carlo algorithms to model the flow of radiation and other physical phenomena
- Willow Garage Inc. (USA): Distributed Rigid Body dynamics in ROS (Robot Operating System) on clusters
- IIJ Innovation Institute Inc. (Japan): network servers handling a massive number of concurrent connections

Work from these organisations is progressing well. The LANL team are well on their way to a demonstrator of a parallel particle tracer which is to be presented to other developers in the laboratory. Dragonfly have done some explatory coding in Haskell (with one working model as a prototype), and have identified some concrete performance needs, particularly a library for sampling random distributions in Haskell. Kazu from IIJ has developed Mighttpd 2 (→ 4.2.7), a web-server on top of WAI/Warp, providing basic web features and CGI.

For the wider community, the tangible outcomes of the project so far have been an MPI binding (→ 4.1.2) now on Hackage, a number of bugfixes to the GHC runtime system, and improvements to c2hs.

Work is now underway to make avaible in Haskell the “Modified Additive Lagged Fibonacci” random number generator and perhaps other RNGs from the C++/Fortran SPRNG library. We currently have a binding to the library which we are using as the basis to test the native Haskell implementation being developed.

Work is also underway to improve tools for profiling parallel Haskell programs. We are extending ThreadScope (and the associated backend infrastructure such as the GHC EventLog) to support profiling of multi-process or distributed Haskell systems such as client/server or MPI programs. Building off this work, we are also adding better profiling for single-process programs by making it possible to compare multiple runs of the same program (e.g., different inputs or slightly different code) on the same timeline. These improvements will be accompanied by ongoing work on adding new visualisations to ThreadScope, for example, the rate of parallel spark creation and the distribution of spark evaluation times.

In addition to the technical outcomes, this project has also resulted in some new resources for the parallel Haskell community. For the various groups working on parallel Haskell, we have established a new public mailing list to encourage collaboration and to improve the visibility of each others' efforts. For newcomers to Haskell and/or parallelism and concurrency, we have developed a Parallel Haskell portal steering users towards the most immediately useful tools and documentation to get started. Finally for the community as a whole, we are putting together a monthly digest to highlight work from people using and developing Parallel Haskell. Further progress on the project will be reported back to the community on these channels.

Further reading

http://haskell.org/haskellwiki/Parallel_GHC_Project

4.2 Haskell and the Web

4.2.1 GHCJS: Haskell to Javascript compiler

Report by:	Victor Nazarov
Status:	0.1.0 released

GHCJS currently is a GHC back-end which produces Javascript code. Modern Javascript environments become more and more advanced. TraceMonkey and V8 engines allow very fast Javascript execution. It is possible, for instance, to create an in-browser hardware emulator: an emulated CPU's instructions are compiled down to Javascript functions, and Javascript instructions are compiled to the native host CPU's instructions by Javascript JIT-compilers (http://weblogs.mozillazine.org/roc/archives/2010/11/implementing_a.html).

The idea to bring the power of the Haskell language to the world of AJAX-applications is not new. It has been proposed many times in Haskell-café. The success of Google's GWT was uncomfortable to watch, when our beloved language lacked such a feature. The first implementation I know is Dmitry Golubovsky's YHC back-end (http://www.haskell.org/haskellwiki/Yhc/Javascript). The second one was my GHC backend hs2js (http://vir.mskhug.ru/). There were differences between the two projects. Dmitry had tried to provide a Haskell environment to develop everything in Haskell. He had developed an automated conversion tool to generate Haskell-bindings from DOM IDL specifications provided by the W3C. My aim was more modest: I thought that we could use Haskell to implement complex logic. The ability to use Parsec in a browser was asked for several times in Haskell-café. With the latter approach we can extend existing Javascript-applications with algorithms implemented in Haskell. UHC (→ 2.4) started to implement a Javascript-backend recently (http://utrechthaskellcompiler.wordpress.com/2010/10/18/haskell-to-javascript-backend/), but I have not looked at it, yet.

GHCJS is a fresh rewrite of hs2js that was started in August 2010. It is currently a standalone tool that uses GHC as a library and produces a `.js`-file for each Haskell-module. Javascript code can load any Haskell-module and evaluate any exported Haskell-value. Some examples that are available with the GHCJS package show some simple Haskell programs like generation of a sequence of prime-numbers. Each Haskell module is currently a standalone Javascript file. When a value of some module is needed, the module is loaded dynamically.

The code is available at the GHCJS github page (see below) under the terms of the BSD3 license. It was tested with GHC 6.12.

There are many tasks awaiting completion with GHCJS:

A faster and more robust module loader: Now it loses a lot of time on 404 errors, trying to access modules in the wrong package directory. I plan to use GHC's package abstraction. A package will be a Web-server's directory and Javascript's namespace. Every module will be unambiguously associated with one package. It will become possible to load a module with one unambiguous HTTP-request. This change will short the loading time of Haskell programs.

Make it work in all major browsers: There are some minor problems with Internet Explorer. But it should be trivial to fix them.

FFI support: FFI support should make the whole thing generally usable. FFI-exports should generate easily-callable Javascript functions that will type-check their arguments to make a combination of dynamically-typed Javascript and statically-typed Haskell seamless. FFI-imports will allow the implementation of DOM-manipulation in Haskell programs.

Further reading

https://github.com/sviperll/ghcjs

4.2.2 WAI

Report by:	Matt Brown
Status:	stable

The Web Application Interface (WAI) is an interface between web applications and web servers. By targeting the WAI, a web application can get access to multiple servers; and through WAI, a server can support web applications never intended to run on it.

WAI has matured significantly since the last HCAR. Much progress has been made in areas of efficiency, generality, and standardization. While WAI is perhaps most often used in conjunction with the Yesod web framework (→ 4.2.8), these gains have helped attract more interest from non-yesod projects. Notable examples include Hoogle, the popular Haskell API search engine (→ 5.2.2), Hums, a UPnP media server, and the Happstack web framework (→ 4.2.6).

Key developments include:

- The release of Warp (→ 4.2.3), a high-performance HTTP backend written in Haskell.
- The adoption of http-types, a standard interface for HTTP servers and clients.
- The adoption of text, an efficient representation for unicode data.

Matt Brown is taking over future maintenance of WAI from Michael Snoyman.

Further reading

http://github.com/snoyberg/wai

4.2.3 Warp

Report by:	Matt Brown

Warp is a high performance, easy to deploy HTTP server backend for WAI (→ 4.2.2). Warp has replaced FastCGI as the officially recommended WAI backend. It evolved out of WAI's SimpleServer after some performance and security tuning by Michael Snoyman and Matt Brown. Due to the combined use of ByteStrings, Blaze-Builder, Enumerators, and GHC's improved I/O manager, Wai+Warp has consistently proven to be one of the most performant web deployment options available. Warp currently services Hoogle (→ 5.2.2), hums, and several production Yesod web sites (→ 4.2.8).

"Warp: A Haskell Web Server" by Michael Snoyman was published in the May/June 2011 issue of IEEE Internet Computing:

- Issue page: http://www.computer.org/portal/web/csdl/abs/mags/ic/2011/03/mic201103toc.htm
- PDF: http://steve.vinoski.net/pdf/IC-Warp_a_Haskell_Web_Server.pdf

4.2.4 Holumbus Search Engine Framework

Report by:	Uwe Schmidt
Participants:	Timo B. Hübel, Sebastian Gauck, Stefan Schmidt, Sebastian Schröder
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 framework supports distributed computations for building indexes and searching indexes. This is done with a MapReduce like framework. The MapReduce framework is independent of the index- and search-components, so it can be used to develop distributed systems with Haskell.

The framework is now separated into four packages, all available on Hackage.

- The Holumbus Search Engine
- The Holumbus Distribution Library
- The Holumbus Storage System
- The Holumbus MapReduce Framework

The search engine package includes the indexer and search modules, the MapReduce package bundles the distributed MapReduce system. This is based on two other packages, which may be useful for their on: The Distributed Library with a message passing communication layer and a distributed storage system.

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
- Git repository containing the current development version of all packages under https://github.com/fortytools/holumbus
- Distributed building of search indexes

Current Work

There are two running projects. The first, a masters thesis done by Sebastian Schröder, deals with the development of a framework for news systems. The functionality will be like with google news, but the target is to build news systems for specialized topics. In the second project a search engine for the FH-Wedel web site will be built. The new aspect in this application is a specialized search for appointments, deadlines, announcements, meetings and other dates.

Further reading

The Holumbus web page (http://holumbus.fh-wedel.de/) includes downloads, Git 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. Sebastian Gauck's thesis dealing with the crawler component is available at http://holumbus.fh-wedel.de/src/doc/thesis-indexing.pdf The thesis of Stefan Schmidt describing the Holumbus MapReduce is available via http://holumbus.fh-wedel.de/src/doc/thesis-mapreduce.pdf.

4.2.5 gitit

Report by:	John MacFarlane
Participants:	Gwern Branwen, Simon Michael, Henry Laxen, Anton van Straaten, Robin Green, Thomas Hartman, Justin Bogner, Kohei Ozaki, Dmitry Golubovsky, Anton Tayanovskyy, Dan Cook, Jinjing Wang
Status:	active development

Gitit is a wiki built on Happstack (→ 4.2.6) and backed by a git, darcs, or mercurial filestore. Pages and uploaded files can be modified either directly via the VCS's command-line tools or through the wiki's web interface. Pandoc (→ 7.2.3) is used for markup processing, so pages may be written in (extended) markdown, reStructuredText, LaTeX, HTML, or literate Haskell, and exported in thirteen different formats, including LaTeX, ConTeXt, DocBook, RTF, OpenOffice ODT, MediaWiki markup, EPUB, and PDF.

Notable features of gitit include:

◦ Plugins: users can write their own dynamically loaded page transformations, which operate directly on the abstract syntax tree.
◦ Math support: LaTeX inline and display math is automatically converted to MathML, using the `texmath` library.
◦ Highlighting: Any git, darcs, or mercurial repository can be made a gitit wiki. Directories can be browsed, and source code files are automatically syntax-highlighted. Code snippets in wiki pages can also be highlighted.
◦ Library: Gitit now exports a library, `Network.Gitit`, that makes it easy to include a gitit wiki (or wikis) in any Happstack application.
◦ Literate Haskell: Pages can be written directly in literate Haskell.

Further reading

http://gitit.net (itself a running demo of gitit)

4.2.6 Happstack

Report by:	Jeremy Shaw

The Happstack project is focused on leveraging the unique characteristics of Haskell to create a web framework which is easier to use, more robust, and more scalable than would be possible in other languages.

The Happstack efforts are divided into three categories:

Integration Where possible, we prefer to integrate and improve existing third party libraries from hackage. For example, we have added integration for many templating libraries such as HSP, Heist, Hamlet, HStringTemplate, BlazeHtml, and digestive functors. In some cases this has included significant improvements and bugfixes to the upstream source code.

Creation In other cases the technology we need does not yet exist and so we implement it ourselves. This includes libraries such as web-routes (for type-safe URL routing) and MACID (a native Haskell persistent data store).

Documentation At the most basic level this includes documenting the API and creating tutorials. But an important long term goal is describing how to architect systems that are scalable, robust, and maintainable.

We are currently focused on the Happstack 7 release which is centered around improvements to MACID. MACID is a persistent data store which can hold arbitrary Haskell data types. Like a traditional database, MACID provides operations which are Atomic, Consistent, Isolated, and Durable. Pull the plug on your server and restart with all your data intact.

MACID is superior to other SQL and NoSQL solutions because it can natively store almost any Haskell datatype. No need to marshal data or give up complex data structures. Queries are written in straight-forward (and powerful) Haskell. No need to learn a special, limited query language or funky DSL.

Compared to previous versions of MACID, the new version is, faster, more robust, and less "magical". But even more importantly, happstack-data and happstack-state have been replaced by two new packages safecopy and acid-state, which are completely independent of the Happstack project. We hope this will encourage more wide spread adoption of MACID by other projects.

We also plan to provide benchmarks of MACID performance compared to other popular solutions like MongoDB, Redis, MySQL, etc.

Future plans

Happstack 8 will migrate to an iteratee-based HTTP backend for even better performance and resource management. The tentative plan is to use Warp.

Happstack 9 will focus on improving some of the higher level components such as authentication and session management.

Further reading

- http://www.happstack.com/
- http://www.happstack.com/docs/crashcourse/index.html
- http://acid-state.seize.it/

4.2.7 Mighttpd2 — Yet another Web Server

Report by:	Kazu Yamamoto
Status:	open source, actively developed

Mighttpd (called mighty) version 2 is a simple but practical Web server in Haskell. It is now working on Mew.org providing basic web features and CGI (mailman and contents search).

Mighttpd version 1 was implemented with two libraries *c10k* and *webserver*. Since GHC 6 uses select(), more than 1,024 connections cannot be handled at the same time. The *c10k* library gets over this barrier with the pre-fork technique. The *webserver* library provides HTTP transfer and file/CGI handling.

Mighttpd 2 stops using the *c10k* library because GHC 7 starts using epoll()/kqueue(). The file/CGI handling part of the *webserver* library is re-implemented as a web application on the *wai* library (→ 4.2.2). For HTTP transfer, Mighttpd 2 links the *warp* library (→ 4.2.3) which can send a file in zero copy manner thank to sendfile().

You can install Mighttpd 2 (*mighttpd2*) from HackageDB.

Further reading

http://www.mew.org/~kazu/proj/mighttpd/en/

4.2.8 Yesod

Report by:	Greg Weber
Participants:	Michael Snoyman
Status:	beta

Yesod is a web framework using the Haskell language to make web programming safer, fast, more productive, and fun. The Haskell community is a very progressive type, and have often approached solving the issues of web development in Haskell with a revolutionary mindset — offering up solutions to web development that use continutaions, components, or recommending new forms of data storage. In contrast, Yesod is in many ways a traditional MVC REST web framework — the revolution comes from consitently applying Haskell's strengths to that model.

Type-safety guarantees against programmer errors such as mis-typing a URL, or forgetting a closing HTML tag, but also allows us to make higher-level security guarantees. We can guarantee against XSS attacks and SQL injections. When type safety conflicts with programmer productivity, Yesod is not afraid to use Haskell's most advanced features of Template Haskell and quasi-quoting to provide easier development for its users. In particular, these are used for declarative routing, declarative schemas, and compile-time templates. Haskell provides us with fast code, and GHC7 provides us with the ability to deploy with a highly-scalable web server that can serve tens of thousands of concurrent users, but is dead simple to deploy (→ 4.2.2).

MVC stands for model-view-controller. The preferred library for models is Persistent (→ 6.4.2), which provides a type-safe interface to data stores of your choosing. Views are handled by the Hamlet family of compile-time template languages. Controllers work with declarative routing to easily provide different response types based on the request.

Instead of providing a single monolithic package, Yesod is broken up into many smaller projects. This means that many of the powerful features of Yesod can be used in your own web development tool stack without issue. Packages for authentication, client-side encrypted session data, middlewares, web encodings, YAML, persistence, HTML templating and more are all fully available on Hackage, without any reliance on Yesod.

Yesod is currently on its 0.8 version. The last HCAR entry was for the 0.5 version. Since then we have added:

- GHC7 support with GHC6 compatibility
- Migration to using the aeson JSON package
- Security — CSRF protection for forms and sessions that can be tied to a static IP address
- Automatic javascript minification
- The Lucius template language — a superset of CSS that adds features such as automatic nesting
- Improved Hamlet (html templating language) syntax — Hamlet syntax is now html by default, there is no need to learn a radical new syntax. Just by making white space significant to remove closing tags, it eliminates the main source of invalid HTML. With that, and some id/class shortcuts, and making attribute quoting optional, it eliminates the main sources of tedium in HTML.
- Preliminary support for coffeescript (a better javascript) templates.
- Atom/RSS feeds
- Improved static file serving — directory listings and beta support for caching headers

We are working towards a 1.0 release by this summer. 1.0 to us means API stability and a web framework that gives developers all the tools they need for productive web development. For the 1.0 release we have the following goals:

- Polish the development environment — we want an ease of development on par with dynamic languages.
- Great documentation, including improved ways for the community to contribute documentation.
- Easier testing for your application.
- Easier interopration with different templating languages.
- Support for faster javascript loading.
- Better form generation.
- A complete i18n (internationalization) solution.

Every major 1.0 feature is already underway, and we already have a productive framework in use by the Haskell community, including commercial users.

To see an example site with source code available, you can view Haskellers (→ 1.1) source code: (https://github.com/snoyberg/haskellers).

The Yesod site (http://yesodweb.com/) is a great place for information. It has code examples, screencasts, the Yesod blog and — most importantly — a book on Yesod. The book is not yet complete, but provides a very solid introduction to the main features, and it is constantly being revised and expanded.

Further reading

http://yesodweb.com/

4.2.9 Snap Framework

Report by:	Doug Beardsley
Participants:	Gregory Collins, Shu-yu Guo, James Sanders, Carl Howells, Shane O'Brien, Ozgun Ataman, Chris Smith
Status:	active development

The Snap Framework is a web application framework built from the ground up for speed, reliability, and ease of use. The project's goal is to be a cohesive high-level platform for web development that leverages the power and expressiveness of Haskell to make building websites quick and easy.

The Snap Framework has seen two major releases (0.3 and 0.4) since the last HCAR with contributions from two new core contributors. The framework was also featured in the January issue of IEEE Internet Computing magazine in their column "The Functional Web". Some of the most significant new features are:

- SSL support
- A development mode that uses Hint (http://haskell.org/communities/11-2008/html/report.html#hint) to recompile your application on the fly when you make code changes
- Support for file uploads and the multipart/form-data content type
- The ability to modify socket timeouts, making it possible to have long-running request handlers
- A new HTML5 parser designed specifically for the Heist template system

We also have development in progress for sessions, authentication, and mongoDB support. These will be available on hackage once design solidifies, but they are usable right now for developers who do not mind working with a less stable code base.

The team is currently working on infrastructure and an API to facilitate modular web development. From that base we plan to build out more of the high-level features that developers have come to expect from modern web frameworks.

Further reading

- IEEE Article: http://steve.vinoski.net/blog/2011/01/21/column-on-the-snap-framework/
- Intro to xmlhtml: http://cdsmith.wordpress.com/2011/02/05/html-5-in-haskell/
- http://snapframework.com

4.2.10 rss2irc

Report by:	Simon Michael
Status:	occasional development; suitable for production use

rss2irc is an IRC bot that polls a single RSS or Atom feed and announces new items to an IRC channel, with options for customizing output and behavior. It aims to be an easy to use, dependable bot that does its job and creates no problems.

rss2irc was published in 2008 by Don Stewart. Simon Michael took over maintainership in 2009, with the goal of making a robust low-maintenance bot to stimulate development in various free/open-source software communities. It is currently used for several full-time bots including:

- hackagebot — announces new hackage releases in #haskell
- hledgerbot — announces hledger commits in #ledger
- zwikicommitbot — announces Zwiki commits in #zwiki
- squeaksobot — announces Squeak and Smalltalk-related Stack Overflow questions in #squeak
- squeakquorabot — announces Squeak/Smalltalk-related Quora questions in #squeak
- etoystrackerbot — announces new Etoys bugs in #etoys

- etoysupdatesbot — announces Etoys commits in #etoys
- planetzopebot — announces new planet.zope.org posts in #zope

The project is available under BSD license from its home page at http://hackage.haskell.org/package/rss2irc.

A 0.5 release containing robustness and feature improvements is in preparation and should arrive soon. More testers and contributors are invited.

Further reading

http://hackage.haskell.org/package/rss2irc

4.3 Haskell and Games

4.3.1 FunGEn

Report by:	Simon Michael
Status:	usable; ready for contributors and users

FunGEn (Functional Game Engine) is a BSD-licensed cross-platform 2D game engine implemented in and for Haskell, using OpenGL and GLUT.

FunGEn was created in 2002 by Andre Furtado, who then moved on to other projects. In 2008 it was updated for current GHC by Simon Michael and then again by Miloslav Raus, who also cabalised it. Earlier this year Simon revived it again, with a GHC 6.12-tested 0.3 release on Hackage, preliminary haddockification and an updated home page.

This is currently probably the easiest way to build a cross-platform graphical game with Haskell, due to its convenient game building framework and its widely-available dependencies (OpenGL and GLUT). FunGEn comes with several working examples that are quite easy to read and build (pong, worms).

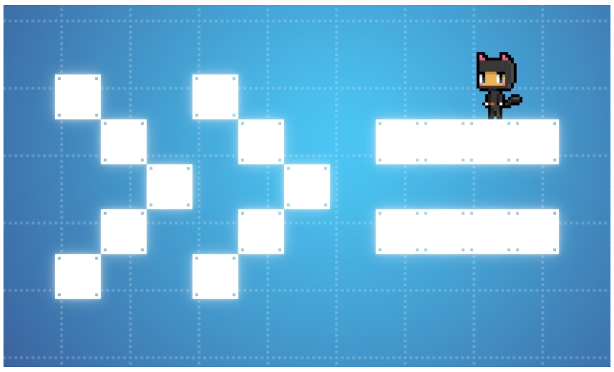
The main Darcs repository is now http://darcsden.com/simon/fungen, which also serves as the project's main home page. Start here to find all code and documentation. See also the #haskell-game channel on irc.freenode.net for discussion.

Further reading

http://darcsden.com/simon/fungen

4.3.2 Nikki and the Robots

Report by:	Sönke Hahn
Participants:	Joyride Laboratories GbR
Status:	alpha, active



Nikki and the Robots is a 2D platformer written in Haskell and produced by Joyride Laboratories. Nikki, the protagonist, walks and jumps around the levels wearing a cute ninja/cat costume. Nikki refrains from using any tools or weapons, with one exception: The Robots. These come in various types with different abilities and can be used by Nikki to solve puzzles, overcome obstacles, and complete the level tasks. The game features an integrated level editor.

We made our first binary release of Nikki and the Robots in April 2011.

Publishing

We are releasing the game and the level editor under an open source license (LGPL). The included graphics are published under a permissive Creative Commons license (cc-by-sa). We are also planning to create a server that will allow players to upload the levels they created and download levels from other players. We hope that a community of coders, level creators, and players will emerge around the game.

Simultaneously, we are working on episodes that we plan to sell via the game. These will include new graphics, more robots, a story line, other characters, and other surprises.

(Just to clarify: The licensing is very permissive. It allows others to create their own episodes and distribute them freely or sell them. This would be very welcome. If anybody is interested in this, we propose to join forces and sell all our episodes through one system.)

Technologies Used

- Qt for user input and rendering.
- OpenGL as an efficient rendering backend for Qt. Everything will remain 2D, though - we promise!
- Hipmunk, the Haskell bindings to the chipmunk physics engine.

Getting Involved

The project is still in alpha stage, so there are some features that are not yet implemented. For some, we have a clear vision on how to implement them; for others, we do not. If you want to get involved, check out our darcs repo, our launchpad site, and do not hesitate to contact us.

Further reading

- http://joyridelabs.de
- http://joyridelabs.de/game/code/
- http://joyridelabs.de/game/download/

4.3.3 Freekick2

Report by:	Antti Salonen
Status:	experimental, active development

See: http://www.haskell.org/communities/05-2010/html/report.html#sect6.11.1.

4.4 Haskell and Compiler Writing

4.4.1 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* (i.e., functions that do to any data type what *foldr* does to lists). You 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.

An AG program is a collection of rules, which are pure Haskell functions between attributes. Idiomatic tree computations are neatly expressed in terms of copy, default, and collection rules. Attributes themselves can masquerade as subtrees and be analyzed accordingly (higher-order attribute). The order in which to visit the tree is derived automatically from the attribute computations. The tree walk is a single traversal from the perspective of the programmer.

Nonterminals (data types), productions (data constructors), attributes, and rules for attributes can be specified separately, and are woven and ordered automatically. These aspect-oriented programming features make AGs convenient to use in large projects.

The system is in use by a variety of large and small projects, such as the Utrecht Haskell Compiler UHC (→2.4), the editor Proxima for structured documents (http://www.haskell.org/communities/05-2010/html/report.html#sect6.4.5), the Helium compiler (http://www.haskell.org/communities/05-2009/html/report.html#sect2.3), the Generic Haskell compiler, UUAG itself, and many master student projects. The current version is 0.9.29 (July 2010), is extensively tested, and is available on Hackage.

We are working on the following enhancements of the UUAG system:

First-class AGs We provide a translation from UUAG to AspectAG (→4.4.2). AspectAG is a library of strongly typed Attribute Grammars implemented using type-level programming. With this extension, we can write the main part of an AG conveniently with UUAG, and use AspectAG for (dynamic) extensions. Our goal is to have an extensible version of the UHC.

Fixpoint evaluation We incorporated a fixed-point evaluation scheme for circular grammars. A cycle is broken by specifying an initial value for an attribute on the cycle, and repeating the evaluation with an updated value until it converges.

Step-wise evaluation We provide the possibility to evaluate AGs step-wise. The evaluation for a nonterminal may yield user-defined progress reports, and we can direct the evaluation until the next progress report. With this mechanism, we can resolve nondeterminism and encode breadth-first search strategies.

Further reading

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

4.4.2 AspectAG

Report by:	Marcos Viera
Participants:	Doaitse Swierstra, Wouter Swierstra
Status:	experimental

AspectAG is a library of strongly typed Attribute Grammars implemented using type-level programming.

Introduction

Attribute Grammars (AGs), a general-purpose formalism for describing recursive computations over data types, avoid the trade-off which arises when building software incrementally: should it be easy to add new data types and data type alternatives or to add new operations on existing data types? However, AGs are usually implemented as a pre-processor, leaving e.g. type checking to later processing phases and making interactive development, proper error reporting and debugging difficult. Embedding AG into Haskell as a combinator library solves these problems. Previous attempts at embedding AGs as a domain-specific language were based on extensible records and thus exploiting Haskell's type system to check the well-formedness of the AG, but fell short in compactness and the possibility to abstract over oft occurring AG patterns. Other attempts used a very generic mapping for which the AG well-formedness could not be statically checked. We present a typed embedding of AG in Haskell satisfying all these requirements. The key lies in using HList-like typed heterogeneous collections (extensible polymorphic records) and expressing AG well-formedness conditions as type-level predicates (i.e., typeclass constraints). By further type-level programming we can also express common programming patterns, corresponding to the typical use cases of monads such as Reader, Writer, and State. The paper presents a realistic example of type-class-based type-level programming in Haskell.

Current Status

In the current version (0.3) we have included support for local and higher-order attributes. Furthermore, a translation from UUAG (→4.4.1) to AspectAG is added to UUAGC as an experimental feature.

Background

The approach taken in AspectAG was proposed by Marcos Viera, Doaitse Swierstra, and Wouter Swierstra in the ICFP 2009 paper "Attribute Grammars Fly First-Class: How to do aspect oriented programming in Haskell".

Further reading

http://www.cs.uu.nl/wiki/bin/view/Center/AspectAG

4.4.3 Berp

Report by:	Bernie Pope
Status:	under development

Berp is an implementation of Python 3. At its heart it is a translator which takes Python code as input and generates Haskell code as output. The Haskell code is fed into a Haskell compiler (GHC) for compilation to machine code or interpretation as byte code. One of the main advantages of this approach is that berp is able to use the rich functionality provided by the GHC runtime system with minimal implementation effort. Berp provides both a compiler and an interactive interpreter, and for the most part it can be used in the same way as CPython (the main Python implementation). Although berp is in the early stages of development, it is able to demonstrate some novel capabilities (compared to CPython), such as tail-call optimisation and call-with-current-continuation.

The syntactic analysis component of berp is provided by a separate Haskell library called language-python (→6.2.2), which can be used independently of berp to produce tools for processing Python source.

Berp underwent a flurry of development activity in the first part of 2010, but since then the pace slowed down as I worked on other projects. Those other projects are now maturing, and I plan to return to berp development soon. Berp is still missing support for some critical features, such as module imports.

Further reading

- http://hackage.haskell.org/package/berp
- http://github.com/bjpop/berp
- http://github.com/bjpop/berp/wiki

4.4.4 LQPL — A Quantum Programming Language Compiler and Emulator

Report by:	Brett G. Giles
Participants:	Dr. J.R.B. Cockett
Status:	v 0.8.4 experimental released

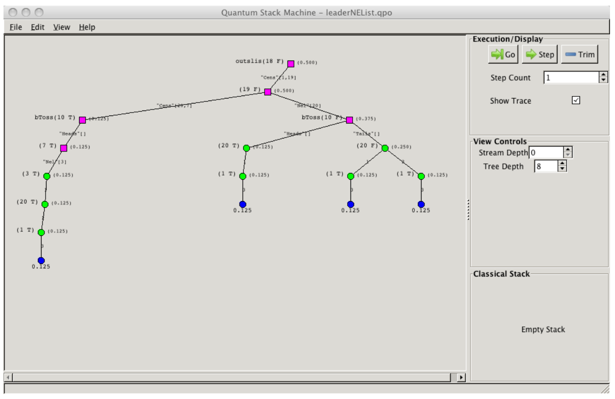
LQPL (Linear Quantum Programming Language) consists of a compiler for a functional quantum programming language and an associated assembler and emulator.

This programming language was inspired by Peter Selinger's paper "Toward a Quantum Programming Language". LQPL incorporates a simplified module / include system (more like C's include than Haskell's import), predefined unitary transforms, quantum control and classical control, algebraic data types, and operations on purely classical data.

Quantum programming allows us to provide a fair coin toss, as shown in the code example below.

```
qdata Coin      = {Heads | Tails}
toss ::( ; c:Coin) =
{  q = |0>;     Had q;
   measure q of
     |0> => {c = Heads}
     |1> => {c = Tails}
}
```

This allows programming of various probabilistic algorithms, such as leader election. The picture below is a screenshot of the emulator part way through leader election, showing a probabilistic list (`outslis`) with equal chances of being one of $[3, 2]$ or $[3, 1]$ and a coin toss (`bToss`) with equal chances of being Heads or Tails.



Step Count 1
Show Trace ✓
Stream Depth 0
Tree Depth 8
Empty Stack

Work on version 0.9 has begun, with the primary goal of further de-coupling the emulator from the user interface. Currently, the user display, the emulator and the assembler are in a monolithic form. Once de-coupled, the intent is to allow the emulator to run independently of the display. This will allow a greater allocation of resources to the emulator, and allow the development of alternate display visualizations.

Further reading

http://pll.cpsc.ucalgary.ca/lqpl/index.html

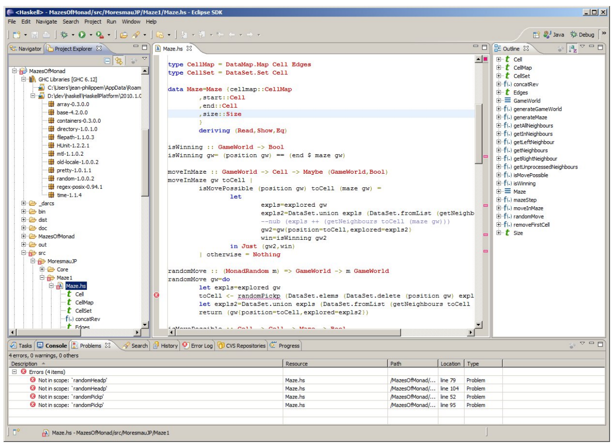
5 Development Tools

5.1 Environments

5.1.1 EclipseFP

Report by:	JP Moresmau
Participants:	B. Scott Michel, Alejandro Serrano, building on code from Thiago Arrais, Leif Frenzel, Thomas ten Cate, and others
Status:	stable, maintained

EclipseFP is a set of Eclipse plugins to allow working on Haskell code projects. It features Cabal integration (.cabal file editor, uses Cabal settings for compilation), and GHC integration. Compilation is done via the GHC API, syntax coloring uses the GHC Lexer. Other standard Eclipse features like code outline, folding, and quick fixes for common errors are also provided. EclipseFP also allows launching GHCi sessions on any module including extensive debugging facilities. It uses Scion to bridge between the Java code for Eclipse and the Haskell APIs. The source code is fully open source (Eclipse License) and anyone can contribute. Current version is 2.0.4, released in March 2011 and supporting GHC 6.12 and 7.0, and more versions with additional features are planned. Feedback on what is needed is welcome! The website has information on downloading binary releases and getting a copy of the source code. Support and bug tracking is handled through Sourceforge forums.



```
type CellMap = DataMap.Map Cell Edges
type CellSet = DataSet.Set Cell

data Maze=Maze {cellmap::CellMap
        ,start::Cell
        ,end::Cell
        ,size::Size
        }
        deriving (Read,Show,Eq)

isWinning :: GameWorld -> Bool
isWinning gw= (position gw) == (end $ maze gw)

moveInMaze :: GameWorld -> Cell -> Maybe (GameWorld,Bool)
moveInMaze gw toCell |
        isMovePossible (position gw) toCell (maze gw) =
                let
                        expls=explored gw
                        expls2=DataSet.union expls (DataSet.fromList (getNeighb
                        --nub (expls ++ (getNeighbours toCell (maze gw)))
                        gw2=gw{position=toCell,explored=expls2}
                        win=isWinning gw2
                in Just (gw2,win)
        | otherwise = Nothing

randomMove :: (MonadRandom m) => GameWorld -> m GameWorld
randomMove gw=do
        let expls=explored gw
        toCell <- randomPickp (DataSet.elems (DataSet.delete (position gw) expl
        let expls2=DataSet.union expls (DataSet.fromList (getNeighbours toCell
        return (gw{position=toCell,explored=expls2})
```

4 errors, 0 warnings, 0 others

Description	Resource	Path	Location	Type
Errors (4 items)				
Not in scope: `randomHeadp'	Maze.hs	/MazesOfMonad/...	line 79	Problem
Not in scope: `randomHeadp'	Maze.hs	/MazesOfMonad/...	line 104	Problem
Not in scope: `randomPickp'	Maze.hs	/MazesOfMonad/...	line 52	Problem
Not in scope: `randomPickp'	Maze.hs	/MazesOfMonad/...	line 95	Problem

Further reading

http://eclipsefp.sourceforge.net/

5.1.2 ghc-mod — Happy Haskell Programming on Emacs

Report by:	Kazu Yamamoto
Status:	open source, actively developed

`ghc-mod` is an enhancement of the Haskell mode on Emacs. It provides the following features:

Completion You can complete a name of keyword, module, class, function, types, language extensions, etc.

Code template You can insert a code template according to the position of the cursor. For instance, "module Foo where" is inserted in the beginning of a buffer.

Syntax check Code lines with error messages are automatically highlighted thanks to flymake. You can display the error message of the current line in another window. `hlint` (→ 5.3.2) can be used instead of GHC to check Haskell syntax.

Document browsing You can browse the module document of the current line either locally or on Hackage.

Function type You can display the type/information of the function on the cursor. (new)

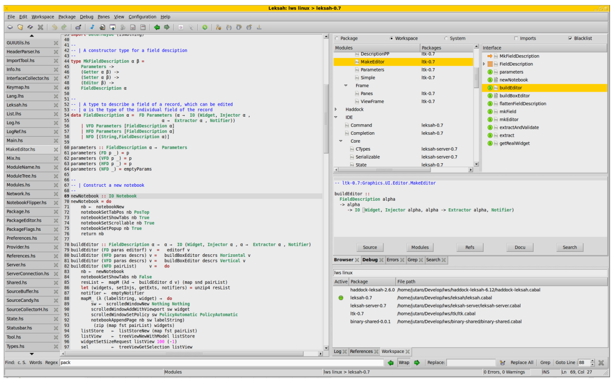
`ghc-mod` consists of code in Emacs Lisp and a subcommand in Haskell. The Emacs code executes the sub-command to obtain information about your Haskell environment. The sub-command makes use of the GHC API for that purpose. `ghc-mod` now supports both GHC 6 and GHC 7.

Further reading

http://www.mew.org/~kazu/proj/ghc-mod/en/

5.1.3 Leksah — The Haskell IDE in Haskell

Report by:	Jürgen Nicklisch-Franken
Participants:	Hamish Mackenzie



```haskell

--
-- | A constructor type for a field desciption
--
type MkFieldDescription α β =
    Parameters ->
    (Getter α β) ->
    (Setter α β) ->
    (Editor β) ->
    FieldDescription α

--
-- | A type to describe a field of a record, which can be edited
-- | α is the type of the individual field of the record
data FieldDescription α =  FD Parameters (α →  IO (Widget, Injector α ,
                                    α →  Extractor α , Notifier))
    | VFD Parameters [FieldDescription α]
    | HFD Parameters [FieldDescription α]
    | NFD [(String,FieldDescription α)]

parameters :: FieldDescription α →  Parameters
parameters (FD p _) = p
parameters (VFD p _) = p
parameters (HFD p _) = p
parameters (NFD _) = emptyParams

--
-- | Construct a new notebook
--
newNotebook :: IO Notebook
newNotebook = do
    nb ←  notebookNew
    notebookSetTabPos nb PosTop
    notebookSetShowTabs nb True
    notebookSetScrollable nb True
    notebookSetPopup nb True
    return nb

buildEditor :: FieldDescription α →  α →  IO (Widget, Injector α , α →  Extractor α , Notifier)
buildEditor (FD paras editorf) v  =   editorf v
buildEditor (HFD paras descrs) v =   buildBoxEditor descrs Horizontal v
buildEditor (VFD paras descrs) v =   buildBoxEditor descrs Vertical v
buildEditor (NFD pairList)     v =   do
    nb ←  newNotebook
    notebookSetShowTabs nb False
    resList ←  mapM (λd →  buildEditor d v) (map snd pairList)
    let (widgets, setInjs, getExts, notifiers) = unzip4 resList
    notifier ←  emptyNotifier
    mapM_ (λ (labelString, widget) →  do
        sw ←  scrolledWindowNew Nothing Nothing
        scrolledWindowAddWithViewport sw widget
        scrolledWindowSetPolicy sw PolicyAutomatic PolicyAutomatic
        notebookAppendPage nb sw labelString)
         (zip (map fst pairList) widgets)
    listStore   ←  listStoreNew (map fst pairList)
    listView    ←  treeViewNewWithModel listStore
    widgetSetSizeRequest listView 100 (-1)
    sel         ←  treeViewGetSelection listView
```

Modules	Packages
DescriptionPP	ltk-0.7
MakeEditor	ltk-0.7
Parameters	ltk-0.7
Simple	ltk-0.7
Frame	
Panes	ltk-0.7
ViewFrame	ltk-0.7
Haddock	
IDE	
Command	leksah-0.7
Completion	leksah-0.7
Core	
CTypes	leksah-server-0.7
Serializable	leksah-server-0.7
State	leksah-0.7

```haskell
-- ltk-0.7:Graphics.UI.Editor.MakeEditor

buildEditor ::
  FieldDescription alpha
  -> alpha
    -> IO (Widget, Injector alpha, alpha -> Extractor alpha, Notifier)
```

lws linux

Active	Package	File path
	haddock-leksah-2.6.0	/home/jutaro/Develop/lws/haddock-leksah-6.12/haddock-leksah.cabal
●	leksah-0.7	/home/jutaro/Develop/lws/leksah/leksah.cabal
	leksah-server-0.7	/home/jutaro/Develop/lws/leksah-server/leksah-server.cabal
	ltk-0.7	/home/jutaro/Develop/lws/ltk/ltk.cabal
	binary-shared-0.0.1	/home/jutaro/Develop/lws/binary-shared/binary-shared.cabal

Leksah is a Haskell IDE written in Haskell. It is still beta quality, but we hope we can publish the 1.0 release this year.

The project has its focus on providing a practical tool for Haskell development. Leksah has already proved its usefulness in industrial projects. We have had positive feedback and are pleased to see that a large number of people are downloading Leksah and we hope you are finding it useful.

Leksah is at a critical point in its development, as it is difficult to bring a project of this size to a success, considering we are just two developers which work on it in their rare spare time. If you can spare some time to work on part of the project, please get in touch by mailing the Leksah group or log onto IRC #leksah. If there is something you do not like about Leksah let us know and we can probably show you where to get started fixing it.

We believe that Leksah can be an important contribution for Haskell, to make its way from an academic language to a valuable tool in industry.

Further reading

http://leksah.org/

5.1.4 HEAT: The Haskell Educational Advancement Tool

Report by:	Olaf Chitil
Status:	active

Heat is an interactive development environment (IDE) for learning and teaching Haskell. Heat was designed for novice students learning the functional programming language Haskell. 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.

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.

A complete re-write of the current version 3.1 is planned to improve the internal structure and make Heat work with GHC.

Further reading

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

5.1.5 HaRe — The Haskell Refactorer

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

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 sixth major release. HaRe supports full Haskell 98, and is integrated with (X)Emacs 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 documentation. Please let us know if you are using the API.

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, PEPM'10, TFP'10, Huiqing's PhD thesis and Chris's PhD thesis). The final report for the project appears there, too.

Recent developments

- HaRe 0.6, which is compatible with GHC-6.12.1, has been released; HaRe 0.6 is available on Hackage, and also downloadable from our project webpage.
- HaRe 0.6 comes with a number of new refactorings, including adding and removing fields and constructors to data-type definitions, folding and unfolding against as-patterns, merging and splitting function definitions, converting between let and where constructs, introducing pattern matching and generative folding.
- Support for automatic detection and semi-automatic elimination of duplicated code in Haskell programs is also available from HaRe 0.6.
- Support for a number of new refactorings for *parallel* Haskell have recently been added to HaRe. These include support to introduce simple divide and conquer parallelism, using the new Strategies module. The refactorings are designed to issue warnings to the user when ill-defined evaluation degrees are set, together with support for adding a threshold value.

Further reading

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

5.2 Documentation

5.2.1 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 and typechecking 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 (→ 5.8.1), 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) as well as the documentation on Hackage.

The latest release is version 2.8.1, released September 3 2010.

Recent changes:

- HTML backend completely rewritten to generate semantically rich XHTML using the xhtml package.
- New default CSS based on the color scheme chosen for the new Haskell wiki, with a pull-out tab for the synopsis.
- Theme engine based on CSS files. Themes can be switched from the header menu.
- Markup support for executable examples/unit-tests.
- Addition of a LaTeX backend.
- Additions and changes to the Haddock API.
- Various smaller new features and bug fixes.

Future plans

- Although Haddock understands many GHC language extensions, we would like it to understand all of them. Currently there are some constructs you cannot comment, like GADTs and associated type synonyms.
- Error messages is an area with room for improvement. We would like Haddock to include accurate line numbers in markup syntax errors.
- On the HTML rendering side we want to make more use of Javascript in order to make the viewing experience better. The frames-mode could be improved this way, for example.
- Finally, the long term plan is to split Haddock into one program that creates data from sources, and separate backend programs that use that data via the Haddock API. This will scale better, not requiring adding new backends to Haddock for every tool that needs its own format.

Further reading

- Haddock's homepage: http://www.haskell.org/haddock/
- Haddock's developer Wiki and Trac: http://trac.haskell.org/haddock
- Haddock's mailing list: haddock@projects.haskell.org

5.2.2 Hoogle

Report by:	Neil Mitchell
Status:	stable

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 plugin.

Hoogle has seen significant revisions in the last few months. Hoogle can now search all of Hackage (→ 5.8.1), and has a brand new look and feel, including instant results as you type. Work continues improving the performance and quality of the results.

Further reading

http://haskell.org/hoogle

5.2.3 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 version 1.14, lhs2TEX has an experimental mode for typesetting Agda code.

The current version is 1.17 and has only minor changes compared to earlier versions, but makes lhs2TEX work with the latest versions of Cabal and ghc. Development has moved to GitHub, which means that there is now also a way to report bugs. Work has started on a more substantial rewrite of lhs2TEX with the goal of cleaning up the internals and making the functionality of lhs2TEX available as a library.

Further reading

- http://www.andres-loeh.de/lhs2tex
- https://github.com/kosmikus/lhs2tex

5.3 Testing and Analysis

5.3.1 shelltestrunner

Report by:	Simon Michael
Status:	occasional development; suitable for daily use

shelltestrunner was first released in 2009, inspired by the test suite in John Wiegley's ledger project. It is a command-line tool for doing repeatable functional testing of command-line programs or shell commands. Tests are defined in one or more files, each test case specifying some or all of: command line, standard input, expected standard output, expected standard error output, expected exit code. Tests can be run selectively or in parallel for greater speed. shelltestrunner is used to test hledger (→ 7.8.7) and at least one other Haskell project.

shelltestrunner is available under the GPL version 3 or later from Hackage or http://joyful.com/repos/shelltestrunner.

The next release will be 1.0, and should appear soon. Feedback and contributors are welcome.

Further reading

http://joyful.com/repos/shelltestrunner

5.3.2 HLint

Report by:	Neil Mitchell
Status:	stable

HLint is a tool that reads Haskell code and suggests changes to make it simpler. For example, if you call `maybe foo id` it will suggest using `fromMaybe foo` instead. HLint is compatible with almost all Haskell extensions, and can be easily extended with additional hints.

There have been numerous feature improvements since the last HCAR, including features to detect duplicated code within a module. HLint can be tried online within hpaste.org.

Further reading

http://community.haskell.org/~ndm/hlint/

5.3.3 hp2any

Report by:	Patai Gergely
Status:	experimental

This project was born during the 2009 Google Summer of Code under the name "Improving space profiling experience". The name hp2any covers a set of tools and libraries to deal with heap profiles of Haskell programs. At the present moment, the project consists of three packages:

- `hp2any-core`: a library offering functions to read heap profiles during and after run, and to perform queries on them.
- `hp2any-graph`: an OpenGL-based live grapher that can show the memory usage of local and remote processes (the latter using a relay server included in the package), and a library exposing the graphing functionality to other applications.
- `hp2any-manager`: a GTK application that can display graphs of several heap profiles from earlier runs.

The project also aims at replacing hp2ps by reimplementing it in Haskell and possibly adding new output formats. The manager application shall be extended to display and compare the graphs in more ways, to export them in other formats and also to support live profiling right away instead of delegating that task to `hp2any-graph`.

The manager application was recently enabled to accept files to load over the command line due to a request. If you feel the need for a feature, do not hesitate to voice it either by e-mailing the author or using the issue tracker of the Google Code repository.

Further reading

- http://www.haskell.org/haskellwiki/Hp2any
- http://code.google.com/p/hp2any/

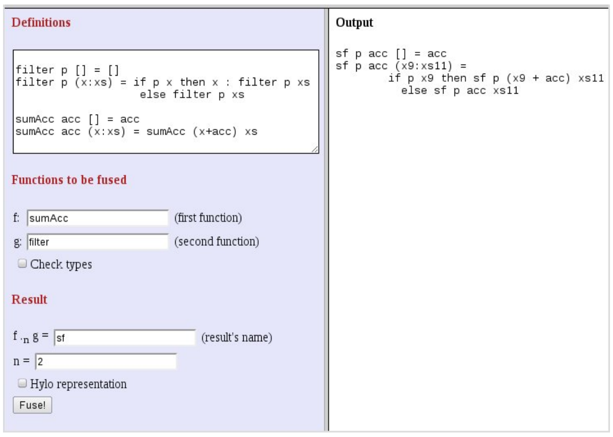
5.4 Optimization

5.4.1 HFusion

Report by:	Facundo Dominguez
Participants:	Alberto Pardo
Status:	experimental

HFusion is an experimental tool for optimizing Haskell programs. The tool performs source to source transformations by the application of a program transformation technique called *fusion*. The aim of fusion is to reduce memory management effort by eliminating the intermediate data structures produced in function compositions. It is based on an algebraic approach where functions are internally represented in terms of a recursive program scheme known as *hylomorphism*.

We offer a web interface to test the technique on user-supplied recursive definitions and since very recently HFusion is also available as a library on Hackage. The user can ask HFusion to transform a composition of two functions into an equivalent program which does not build the intermediate data structure involved in the composition.



```
filter p [] = []
filter p (x:xs) = if p x then x : filter p xs
                  else filter p xs

sumAcc acc [] = acc
sumAcc acc (x:xs) = sumAcc (x+acc) xs
```

f: sumAcc (first function)
g: filter (second function)
☐ Check types

f ·n g = sf (result's name)
n = 2
☐ Hylo representation

```
sf p acc [] = acc
sf p acc (x9:xs11) =
      if p x9 then sf p (x9 + acc) xs11
        else sf p acc xs11
```

In its current state, HFusion is able to fuse compositions of general recursive functions, including primitive recursive functions (like dropWhile or insertions in binary search trees), functions that make recursion over multiple arguments like zip, zipWith or equality predicates, mutually recursive functions, and (with some limitations) functions with accumulators like foldl. In general, HFusion is able to eliminate intermediate data structures of regular data types (sum-of-product types plus different forms of generalized trees).

The next immediate steps will be improving usability of the HFusion library API and working on finding fusable compositions within programs automatically.

Further reading

- HFusion publications: http://www.fing.edu.uy/inco/proyectos/fusion
- HFusion web interface: http://www.fing.edu.uy/inco/proyectos/fusion/tool
- HFusion on Hackage: http://hackage.haskell.org/package/hfusion

5.4.2 Optimizing Generic Functions

Report by:	José Pedro Magalhães
Participants:	Johan Jeuring, Andres Löh
Status:	actively developed

Datatype-generic programming increases program reliability by reducing code duplication and enhancing reusability and modularity. Several generic programming libraries for Haskell have been developed in the past few years. These libraries have been compared in detail with respect to expressiveness, extensibility, typing issues, etc., but performance comparisons have been brief, limited, and preliminary. It is widely believed that generic programs run slower than hand-written code.

At Utrecht University we are looking into the performance of different generic programming libraries and how to optimize them. We have confirmed that generic programs, when compiled with the standard optimization flags of the Glasgow Haskell Compiler (GHC), are substantially slower than their hand-written counterparts. However, we have also found that advanced optimization capabilities of GHC, such as inline pragmas and rewrite rules, can be used to further optimize generic functions, often achieving the same efficiency as hand-written code.

We are continuing our research in this topic and hope to provide more information in the near future.

Further reading

http://dreixel.net/research/pdf/ogie.pdf

5.5 Boilerplate Removal

5.5.1 A Generic Deriving Mechanism for Haskell

Report by:	José Pedro Magalhães
Participants:	Atze Dijkstra, Johan Jeuring, Andres Löh, Simon Peyton Jones
Status:	actively developed

Haskell's deriving mechanism supports the automatic generation of instances for a number of functions. The Haskell 98 Report only specifies how to generate instances for the Eq, Ord, Enum, Bounded, Show, and Read classes. The description of how to generate instances is largely informal. As a consequence, the portability of instances across different compilers is not guaranteed. Additionally, the generation of instances imposes restrictions on the shape of datatypes, depending on the particular class to derive.

We have developed a new approach to Haskell's deriving mechanism, which allows users to specify how to derive arbitrary class instances using standard datatype-generic programming techniques. Generic functions, including the methods from six standard Haskell 98 derivable classes, can be specified entirely within Haskell, making them more lightweight and portable.

We have implemented our deriving mechanism together with many new derivable classes in the Utrecht Haskell Compiler (→2.4). Currently we are working on implementing it in GHC as well, replacing the existing (but rarely used) generic classes. The underlying library for generic data representation, `generic-deriving`, is available on Hackage.

The new mechanism will allow users to define their own generic classes, making instantiation much simpler. Consider enumeration:

```
class GEnum a where
  genum :: [a]
  default genum :: (Representable0 a,
                    Enum' (Rep0 a))
                 ⇒ [a]
  genum = map to0 enum'
```

The *Enum′* and *GEnum* classes are defined by the generic library writer. The end user can then give instances for his/her datatypes without defining an implementation:

```
instance (GEnum a) ⇒ GEnum (Maybe a)
instance (GEnum a) ⇒ GEnum [a]
```

These instances are empty, and therefore use the (generic) default implementation. This is as convenient as writing **deriving** clauses, but allows the user to define more generic classes. This implementation relies on the new functionality of *generic default methods*, like *genum* above, which are like standard default methods but allow for a different type signature. We hope to have a prototype version of the new generic deriving mechanism included in the upcoming GHC 7.2 release.

Further reading

http://dreixel.net/research/pdf/gdmh.pdf

5.5.2 Derive

Report by:	Neil Mitchell
Status:	v2.3.0

See: http://www.haskell.org/communities/05-2010/html/report.html#sect4.1.5.

5.6 Code Management

5.6.1 Darcs

Report by:	Eric Kow
Participants:	darcs-users list
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.

Our most recent major release, Darcs 2.5, was in November 2010. It provides faster repository-local operations, and faster record with long patch histories, among other bug fixes and features.

We are now looking forward to the release of Darcs 2.8 this summer, including Alexey Levan's 2010 Google Summer of Code work on optimised darcs get (using the "optimize –http" command) and a few refinements to Adolfo Builes' cache reliability work. The Darcs 2.8 release is planned to include a faster and more human-readable annotate command and potentially an experimental rebase feature.

In addition to the upcoming release, we are excited to be participating in Google Summer of Code 2011. We have two projects this year, one to develop a bidirectional bridge between Darcs and Git (and potentially other VCSs), and the other to do some new exploratory work on primitive patch types for a future Darcs 3. The bridge project will improve collaboration between Darcs and Git users, allowing each to contribute to projects hosted in the other's VCS of choice. The primitive patches work will allow us to implement some ideas we have been discussing in the Darcs team in recent months, in particular, separation of file dentifiers from file names and the separation of on-disk patch contents from their in-memory representation. Making a prototype implementation of these ideas will give us a better idea how feasible they are in practice and help us to identify the technical difficulities that may be lurking around the corner.

The two projects are very exciting, and they represent a potential trend in Darcs away from catching up with day-to-day issues and towards a more long-term perspective. Meanwhile, we still have a lot progress to make and are always open to contributions. Haskell hackers, we need your help!

Darcs is free software licensed under the GNU GPL. Darcs is a proud member of the Software Freedom Conservancy, a US tax-exempt 501(c)(3) organization. We accept donations at http://darcs.net/donations.html.

Further reading

http://darcs.net

5.6.2 ipatch

Report by:	Joachim Breitner
Status:	working

ipatch brings some of Darcs' specialities, most notably the hunk selection and editing interface, to those who work with plain patch files outside any version control system. Currently, it allows you to interactively and selectively apply a patch or to split a patch into several patch files.

ipatch has not seen a lot of use yet and certainly has rough edges. It can nevertheless be useful already. It can be installed from hackage, and patches are, as always, welcome.

Further reading

- http://hackage.haskell.org/package/ipatch
- https://www.joachim-breitner.de/blog/archives/425-ipatch,-the-interactive-patch-editor.html

5.6.3 DarcsWatch

Report by:	Joachim Breitner
Status:	working

DarcsWatch is a tool to track the state of Darcs (→ 5.6.1) 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.

DarcsWatch continues to be used by the xmonad project (→ 7.8.3), the Darcs project itself, and a few developers. At the time of writing, it was tracking 42 repositories and 4069 patches submitted by 203 users.

Further reading

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

5.6.4 darcsden

Report by:	Simon Michael
Participants:	Alex Suraci
Status:	suitable for casual use; low development activity

http://darcsden.com is a free Darcs (→ 5.6.1) repository hosting service, similar to `patch-tag.com` or (in essence) `github`. The darcsden software is also available (on darcsden) so that anyone can set up a similar service. darcsden was created by Alex Suraci.

The last Hackage release was in 2010. Alex is keeping the existing service running, but has (mostly) moved on to other projects. It is a viable hosting option for smaller projects, with occasional outages/glitches.

The software is available under BSD license.

Further reading

http://darcsden.com

5.6.5 darcsum

Report by:	Simon Michael
Participants:	Dave Love, Simon Marlow
Status:	occasional development; suitable for daily use

darcsum is an emacs add-on providing an efficient, pcl-cvs-like interface for the Darcs revision control system (→ 5.6.1). It is especially useful for reviewing and recording pending changes.

Simon Michael took over maintainership in 2010, and tried to make it more robust with current Darcs. The tool remains slightly fragile, as it depends on Darcs' exact command-line output, and needs updating when that changes. Dave Love has contributed a large number of cleanups.

darcsum is available under the GPL version 2 or later from http://joyful.com/repos/darcsum.

It is due for a release, which will be 1.3. A new maintainer for darcsum would be welcome — please contact Simon Michael ⟨simon@joyful.com⟩ if interested.

Further reading

http://joyful.com/repos/darcsum/

5.6.6 Improvements to Cabal's Test Support

Report by:	Thomas Tuegel
Participants:	Johan Tibell (Mentor)
Status:	active development

As part of the Google Summer of Code 2010, Cabal's test support was improved to allow automated testing of packages. The intent is to provide the technical enhancements necessary for wide adoption of automatic testing in Haskell software, improving the software's general quality. The results of the Summer of Code project were presented at the 2010 Haskell Implementors Workshop, but work is ongoing.

A basic test interface allowing package authors to specify standalone test executables in their package description files will be available in Cabal 1.10 (→ 5.8.1). Cabal can run these tests from the command line and report the aggregate results of the test suite in human- and machine-readable format. Work is in progress to support a standard interface for modules containing multiple test cases; this will make it possible for Cabal to report on the results of individual cases within a test suite. Future work on Hackage will make test reports for uploaded packages available automatically.

Further reading

- http://cabaltest.blogspot.com
- http://haskell.org/haskellwiki/HaskellImplementorsWorkshop/2010

5.6.7 `cab` — A Maintenance Command of Haskell Cabal Packages

Report by:	Kazu Yamamoto
Status:	open source, actively developed

`cab` is a MacPorts-like maintenance command of Haskell cabal packages. Some parts of this program are a wrapper to `ghc-pkg`, `cabal`, and `cabal-dev`.

If you are always confused due to inconsistency of `ghc-pkg` and `cabal`, or if you want a way to check all outdated packages, or if you want a way to remove outdated packages recursively, this command helps you.

Further reading

http://www.mew.org/~kazu/proj/cab/en/

5.6.8 Hackage-Debian

Report by:	Marco Gontijo
Status:	unconcluded

Hackage-Debian is a tool for creating a Debian repository with all, or almost all, of the packages in Hackage. It is highly based on the debian available at http://hackage.haskell.org/package/debian. It should build a snapshot of the Hackage database and then track each new package added to build it on demand. It is still under development, but the first release should be announced soon.

A limitation of the first version being developed is that it only builds the latest version of each library. So, if a library depends on an older version of another library, it will not be built. This is the reason why it does not build all packages, but almost all of them.

Also, the first version will only deal with libraries, but there are plans to also build programs.

The darcs repository for both hackage-debian and the modified version of the debian package that it uses are available at http://marcot.eti.br/darcs/hackage-debian and http://marcot.eti.br/darcs/haskell-debian.

5.7 Interfacing to other Languages

5.7.1 HSFFIG

Report by:	Dmitry Golubovsky
Status:	release

Haskell FFI Binding Modules Generator (HSFFIG) is a tool which parses C include files (`.h`) and generates Haskell Foreign Functions Interface import declarations for all functions, suitable `#define`'d constants, enumerations, and structures/unions (to access their members). It is assumed that the GNU C Compiler and Preprocessor are used. Auto-generated Haskell modules may be imported into applications to access foreign libraries' functions and variables in type-safe manner.

In the current version 1.1.3, speed of processing `#define`'d constants is considerably improved by using HSFFIG's own C language syntax parser to determine suitability of constants for FFI import. Previous versions of HSFFIG invoked an external C compiler for this purpose.

Further reading

- The HSFFIG package on Hackage
  http://hackage.haskell.org/package/HSFFIG
- The HSFFIG Tutorial
  http://www.haskell.org/haskellwiki/HSFFIG/Tutorial
- The FFI Imports Packaging Utility
  http://www.haskell.org/haskellwiki/FFI_imports_packaging_utility

5.8 Deployment

5.8.1 Cabal and Hackage

Report by:	Duncan Coutts

Background

Cabal is the standard package system for Haskell software. It specifies a standard way in which Haskell libraries and applications can be packaged so that it is easy for consumers to use them, or re-package them, regardless of the Haskell implementation or installation platform.

Hackage is a distribution point for Cabal packages. It is an online database of Cabal packages which can be 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

Cabal-1.10 and cabal-install-0.10 were released recently. They are available from hackage and are also included in the current release of the Haskell Platform (→ 2.1).

The major new feature for Cabal-1.10 is "cabal test". This is a feature to allow packages to define test suites. There is an interface to allow build agents such as `cabal` or a hackage buildbot to run the testsuites and collect the results.

The biggest new feature in cabal-install-0.10 is that it can now work with a wider range of targets. Rather than just the package in the local directory or named packages from hackage, it can now work with packages in local directories and local and remote package tarballs. For example:

```
\$ cabal install  ./  ./deps/pkgA/
   ./deps/pkgB-1.0.tar.gz
\$ cabal install
   http://example.com/pkgC-1.0.tar.gz
```

This enables a collection of inter-dependent local packages to be installed in one go. The support for remote tarballs will enable developers to publish beta versions and development snapshots in an ad-hoc way using their own web hosting. This should reduce the pressure to publish betas on hackage.

Looking forward

We have two GSoC students working on Cabal this summer. Mikhail Glushenkov will be working on parallel builds: initially building independent packages in parallel, and if time allows, replacing the serial `ghc --make` with parallel invocations of `ghc -c` on individual modules. Sam Anklesaria will be implementing "cabal repl" which will launch an interactive session (i.e. GHCi) but with all the appropriate pre-processing and context from the project's `.cabal` file.

Progress on the new `hackage-server` has been slow in the last 6 months. Volunteering to help with this would be a great service to the community.

There are of course many improvements we want to make to Cabal, cabal-install, and Hackage. We have had a number of good contributions in the last few months, and we have cleared the previous backlog of patches. As ever our limiting factor remains the amount of volunteer development time and maintainer oversight. We would like to encourage people considering contributing to join the `cabal-devel` mailing list so that we can increase development discussion and improve collaboration. The bug tracker is well maintained and it should be relatively clear to new contributors what is in need of attention and which tasks are considered relatively easy.

Further reading

- Cabal homepage: http://www.haskell.org/cabal
- Hackage package collection: http://hackage.haskell.org/
- Bug tracker: http://hackage.haskell.org/trac/hackage/

5.8.2 Hackage 2.0

Report by:	Matthew Gruen
Participants:	Duncan Coutts
Status:	in development

Hackage 2.0 is a rewrite of the original Hackage (→ 5.8.1) infrastructure intended to provide additional features and better handle Haskell's sustained growth. It was developed to a near-deployable state as part of the 2010 Google Summer of Code program. Enhancing Hackage's role as a package repository, it adds metrics for packages, means of communication between end-users and maintainers, and tools to aid quality assurance.

Currently, Hackage runs an Apache instance to store packages. It is very stable, but also difficult to extend. Plain text files are used to store information, so some features which require plenty of in-memory data manipulation are costly. The new codebase, called hackage-server (in contrast to the current hackage-scripts), uses the Happstack web framework (→ 4.2.6) for just about everything. It employs happstack-state to store native Haskell datatypes in-memory, with a separate file store for the package tarballs themselves.

Features

The primary design goal of hackage-server is to provide a modular, extensible infrastructure for any conceivable feature that might be added to Hackage. It has full feature parity with hackage-scripts and more, with a RESTful backend supporting multiple content formats.

Reverse dependencies, editable tags, and download counts have all been implemented to help locate useful libraries out of thousands. Deprecation, user-submitted build reports, and a user groups system are intended to make maintainance easier. There is also the ability to post packages on a beta-testing index before publishing it on Hackage proper.

Roadmap

The eventual goal is to have the hackage-server codebase serving packages at http://hackage.haskell.org. It is much closer to this now than half a year ago! Further work involves improving performance in both time and memory.

The first major deployment will be a simple mirror for the main Hackage on the sparky server (at sparky.haskell.org, port 8080) which Cabal can set as a remote repository. Afterwards, the mirror will be open for editing by anyone with an account on the main Hackage. The full switchover will occur as soon as we are confident about the stability.

Further reading

◦ Wiki documentation: http://hackage.haskell.org/trac/hackage/wiki/HackageDB/2.0
◦ Code: `darcs get` http://code.haskell.org/hackage-server

5.8.3 Capri

Report by:	Dmitry Golubovsky
Status:	experimental

Capri (abbreviation of **CA**bal **PRI**vate) is a wrapper program on top of cabal-install to operate it in project-private mode. In this mode, there is no global or user package databases; only one package database is defined, private to the project, located under the root directory of a project.

Capri invokes cabal-install and ghc-pkg in the way that only a project's private package database is visible to them. Starting with a minimally required set of packages, all necessary dependencies will be installed per project, not affecting user or global databases. This helps maintain a clean build environment without the risk of accidental installation of conflicting versions of the same package which sometimes happens with the global packages database.

Capri is mainly intended to build executable programs. It depends on certain features of GHC, and is not usable with other Haskell compilers.

Further reading

◦ The Capri package on Hackage
http://hackage.haskell.org/package/capri
◦ The Capri Tutorial
http://www.haskell.org/haskellwiki/Capri

5.8.4 Shaker

Report by:	Anthonin Bonnefoy
Status:	active development

Shaker is an interactive build tool which allows to compile and execute tests on a Haskell project and provides several features like:

◦ Continuous mode: In continuous mode, an action (compile or test) is triggered by source changes.

◦ Automatic test discovery: Shaker discovers and executes tests via the GHC API. All exported QuickCheck properties and HUnit test cases can be executed by Shaker.

◦ Selectable test execution: One or several tests can be selected for execution using regular expressions.

◦ `test-framework` integration for test execution

◦ Easy configuration: Shaker reuses cabal configuration so there is no need for extra configuration if your project is already cabalized.

Shaker can be used to type check your code as you edit it; With the `~compile` command, a compilation will be executed as soon as a source change is detected. You can also execute a specific test on source change with `~test aTestName`.

Future plans

Shaker is incompatible with several projects due to some special cases not managed, and current development aims to make it compatible with more projects. After this, the next feature will be the possibility to execute only previously failing tests.

Further reading

◦ http://hackage.haskell.org/package/shaker
◦ http://github.com/bonnefoa/Shaker

6 Libraries

6.1 Processing Haskell

6.1.1 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 (http://www.haskell.org/communities/05-2009/html/report.html#sect2.3) 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. This work has been continued by Mathijs Swint, but the results of his work have told us that although we do have a lot of data, we need quite a bit more in order to get significant results from Neon.

6.1.2 mueval

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

See: http://www.haskell.org/communities/05-2010/html/report.html#sect5.3.3.

6.2 Parsing and Transforming

6.2.1 The grammar-combinators Parser Library

Report by:	Dominique Devriese
Status:	partly functional

The grammar-combinators library is an experimental next-generation parser library written in Haskell (LGPL license). The library features much of the power of a parser generator like Happy or ANTLR, but with the library approach and most of the benefits of a parser combinator library.

The project's initial release was in September 2010. A paper about the main idea is being published at the PADL'11 conference and an accompanying technical report with more implementation details is available online. The library is published on Hackage under the name grammar-combinators.

We believe this library is an ideal place for innovations in practical functional parsing libraries. The library adds substantial fundamental power to traditional parser combinator libraries and opens up the path to implementing many parsing techniques that were previously impossible. We believe people interested in parsing techniques will find the library ideal for implementing their ideas and we encourage all contributions.

However, the library still needs a lot of love before it is suited for mainstream use. Performance is not ideal and many real-world features are missing. People interested to work on these topics are very welcome to contact us!

Further reading

http://projects.haskell.org/grammar-combinators/

6.2.2 language-python

Report by:	Bernie Pope
Status:	stable

Language-python is a Haskell library for lexical analysis, parsing, and pretty printing Python code. It supports versions 2.x and 3.x of Python. The parser is implemented using the happy parser generator, and the alex lexer generator. It supports source accurate span information and optional parsing of comments. A separate package called language-python-colour is available on Hackage which demonstrates the use of the library to render Python source in coloured XHTML. the library is also used for the syntactic analysis component of the berp Python compiler (→ 4.4.3).

Further reading

- http://hackage.haskell.org/package/language-python
- http://github.com/bjpop/language-python

6.2.3 Loker

Report by:	Roman Cheplyaka
Participants:	Alexander Batischev
Status:	in development

Loker is a collection of programs to deal with UNIX Shell scripts. It will include a parser, a static analysis tool, and a compiler. The distinctive feature of the project is strong compliance to the POSIX standard. All the parts are written in Haskell.

Currently the main focus is on the correctness of the parser and the quality of its error messages. The work has led to several patches to the Parsec 3 library, which improve its error reporting.

Further reading

http://github.com/feuerbach/loker

6.2.4 epub-metadata

Report by:	Dino Morelli
Status:	stable, actively developed

Library for parsing and manipulating ePub files and OPF package data. An attempt has been made here to very thoroughly implement the OPF Package Document specification.

epub-metadata is available from Hackage, the Darcs repository below, and also in binary form for Arch Linux through the AUR.

See also epub-tools (→ 7.8.10).

Further reading

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

6.2.5 ChristmasTree

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

See: http://haskell.org/communities/05-2009/html/report.html#sect5.5.7.

6.2.6 First Class Syntax Macros

Report by:	Marcos Viera
Participants:	Doaitse Swierstra, Atze Dijkstra, Arthur Baars
Status:	experimental

See: http://haskell.org/communities/05-2010/html/report.html#sect5.4.2.

6.2.7 Utrecht Parser Combinator Library: uu-parsinglib

Report by:	Doaitse Swierstra
Status:	actively developed

The previous extension for recognizing merging parsers was generalized so now any kind of applicative and monadic parsers can be merged in an interleaved way. As an example take the situation where many different programs write log entries into a log file, and where each log entry is uniquely identified by a transaction number (or process number) which can be used to distinguish them. E.g., assume that each transaction consists of an a, a b and a c action, and that a digit is used to identify the individual actions belonging to the same transaction; the individual transactions can now be recognized by the parser:

$$\begin{aligned}
&\mathit{pABC} :: \mathit{Grammar\ String}\\
&\mathit{pABC} = (\lambda a\ d \rightarrow d : a) \mathbin{<\$>} \mathit{pA} \mathbin{<\!\!*\!\!>} (\mathit{pDigit'} \gg\!=\\
&\qquad\qquad \lambda d \rightarrow \mathit{pB} \ \mathbin{*\!\!>} \mathit{mkGram}\ (\mathit{pSym}\ d) \ \mathbin{*\!\!>}\\
&\qquad\qquad\qquad\quad \mathit{pC} \ \mathbin{*\!\!>} \mathit{mkGram}\ (\mathit{pSym}\ d)\\
&\qquad\qquad)
\end{aligned}$$

Now running many merged instances of this parser on the input returns the list of first lines prefixed by their number:

```
run (pmMany(pABC)) "a2a1b1b2c2a3b3c1c3"
 Result: ["2a","1a","3a"]
```

Furthermore the library was provided with many more examples in two modules in the *Demo* directory.

Features

- Much simpler internals than the old library (http://haskell.org/communities/05-2009/html/report.html#sect5.5.8).
- Combinators for easily describing parsers which produce their results online, do not hang on to the input and provide excellent error messages.
- Parsers "correct" the input such that parsing can proceed when an erroneous input is encountered.
- The library provides both an applicative interface and a monadic interface.
- No need for *try*-like constructs which makes writing `Parsec` based parsers tricky.
- Scanners can be switched dynamically, so several different languages can occur intertwined in a single input file.
- Parsers can be run in an interleaved way, thus generalizing the merging and permuting parsers into a single applicative ineterface.

Future plans

The next version will contain a check for grammars being not left-recursive, thus taking away the only remaining source of surprises when using parser combinator libraries. This makes the library great for teaching environments too. Future versions of the library, using even more abstract interpretation, will make use of computed look-ahead information to speed up the parsing process further. Gradually software from Utrecht will be moving to use the new library `uu-parsinglib`.

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⟩. There is a low volume, moderated mailing list at ⟨parsing@cs.uu.nl⟩. More information can be found at http://www.cs.uu.nl/wiki/bin/view/HUT/ParserCombinators.

6.2.8 Regular Expression Matching with Partial Derivatives

Report by:	Martin Sulzmann
Participants:	Kenny Zhuo Ming Lu
Status:	stable

We have substantially improved the performance of our matching algorithms. The latest implementation can be downloaded via hackage. A paper describing our approach has been submitted to ICFP 2011.

Further reading

- http://hackage.haskell.org/package/regex-pderiv
- http://sulzmann.blogspot.com/2010/04/regular-expression-matching-using.html

6.3 Mathematical Objects

6.3.1 normaldistribution: Minimum Fuss Normally Distributed Random Values

Report by:	Björn Buckwalter
Status:	stable

Normaldistribution is a new package that lets you produce normally distributed random values with a minimum of fuss. The API builds upon, and is largely analogous to, that of the Haskell 98 Random module (more recently *System.Random*). Usage can be as simple as: *sample* ← *normalIO*. For more information and examples see the package description on Hackage.

Further reading

http://hackage.haskell.org/package/normaldistribution

6.3.2 dimensional: Statically Checked Physical Dimensions

Report by:	Björn Buckwalter
Status:	active, stable core with experimental extras

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 within the frame of the SI. Example:

$$\begin{aligned}&d :: \mathit{Fractional}\ a \Rightarrow \mathit{Time}\ a \rightarrow \mathit{Length}\ a\\&d\ t = a\ /\ \_2 * t\ \hat{}\ \mathit{pos2}\\&\quad \textbf{where}\ a = 9.82 *\tilde{}\ (\mathit{meter}\ /\ \mathit{second}\ \hat{}\ \mathit{pos2})\end{aligned}$$

The dimensional library is stable with units being added on an as-needed basis. The primary documentation is the literate Haskell source code. The wiki on the project web site has several usage examples to help with getting started.

Ongoing experimental work includes:

- Support for user-defined dimensions and a proof-of-concept implementation of the CGS system of units.

- dimensional-vectors — a rudimentary linear algebra library which statically tracks the sizes of vectors and matrices as well as the physical dimensions of their elements on a per element basis, disallowing non-sensical operations. This library makes it very difficult to accidentally implement, e.g., a Kalman filter incorrectly. My work on dimensional-vectors is need-driven and tends to occur in spurts.

- dimensional-experimental — a library in heavy flux of which the most interesting feature is probably automatic differentiation of functions involving physical quantities. Example:

$$\begin{aligned}&v :: \mathit{Fractional}\ a \Rightarrow \mathit{Time}\ a \rightarrow \mathit{Velocity}\ a\\&v\ t = \mathit{diff}\ \ d\ t\end{aligned}$$

The core library, dimensional, can be installed off Hackage using cabal. The experimental packages can be cloned off of Github.

Dimensional relies on *numtype* for type-level integers (e.g., *pos2* in the above example), *ad* for automatic differentiation, and *HList* (→6.4.1) for type-level vector and matrix representations.

Further reading

- http://dimensional.googlecode.com
- https://github.com/bjornbm/dimensional-vectors
- https://github.com/bjornbm/dimensional-experimental

6.3.3 AERN-Real and Friends

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

AERN stands for Approximating Exact Real Numbers. We are developing a family of libraries that will provide:

- a reliable and fast arbitrary precision correclty rounded **interval arithmetic**, including both standard and inverted intervals with Kaucher arithmetic
- arbitrary precision arithmetic of **polynomial intervals** to
  - automatically reduce overestimations in interval computations
  - efficiently support validated numerical integration
- a type class hierarchy for validated and exact computation, featuring
  - standard mathematical structures such as posets and lattices extended to take account of rounding errors and partially decided relations such as equality
  - separate treatment of numerical order and interval refinement order
  - ability to increase computational effort to reduce the effect of rounding and partiality, converging to zero with infinite effort
  - extensive set of QuickCheck properties for each type class, enabling automatic checking of, e.g., algebraic properties such as associativity extended to take rounding into account
- a framework for distributed dataflow exact numerical computation with tidy exact semantics based on Domain Theory

There are stable older versions of the libraries on Hackage but these lack the type classes described above. We are currently redesigning and rewriting the libraries from scratch, with an imminent release of an interval arithmetic with Double endpoints. A release supporting MPFR endpoints should follow in the summer of 2011 and polynomial arithmetic with an efficient core written in C is also being developed. All development is open and we welcome contributions.

Further reading

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

6.3.4 hmatrix

Report by:	Alberto Ruiz
Participants:	Vivian McPhail
Status:	stable, maintained

`hmatrix` is a purely functional interface to numerical linear algebra, internally implemented using GSL, BLAS, and LAPACK. Recent changes include using by default Data.Vector.Storable from Roman Leshchinskiy's vector package. Future work includes a possible separation of the library into smaller packages.

Further reading

http://perception.inf.um.es/hmatrix

6.4 Data Types and Data Structures

6.4.1 HList — A Library for Typed Heterogeneous Collections

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

See: http://www.haskell.org/communities/05-2010/html/report.html#sect5.6.1.

6.4.2 Persistent

Report by:	Greg Weber
Participants:	Michael Snoyman
Status:	stable

Persistent is a universal, type-safe data store interface for Haskell. Haskell has many different database bindings available. However, most of these have little knowledge of a schema and therefore do not provide useful static guarantees. They also force database-dependent interfaces and data structures on the programmer. Haskellers have attempted a more revolutionary route of creating Haskell specific data stores to get around these flaws. This allows one to easily store any Haskell type, and a great option for certain use cases. However, they constrain one to the storage techniques provided by the library, do not interface well with other languages, and may not have easy and efficient techniques for querying data. In contrast, Persistent allows us to choose among existing databases that are highly tuned for different data storage use cases, to interoperate with other programming languages, and to use a safe, but highly productive query interface.

Recently, Persistent has seen 2 major feature additions of joins and MongoDB support.

Persistent was always designed with newer databases (NoSQL) in mind, but only a Sqlite and Postgresql backend were implemented. The lastest release adds alpha support for MongoDB, which really showcases the advantages of Persistent. By default, MongoDB is schema-less, and the Haskell driver supports this as it should because some use cases can take advantage of this. However, for most use cases there is a known schema, and you are always a typo away from inserting the wrong key or querying a key that does not exist. With Persistent you know at compile time that this will not occur. You also get automatic conversion from the driver type to a normal Haskell data type.

This release also adds application level joins for all backends and SQL joins for the SQL backends.

Future plans

There are 3 main directions for Persistent:

- Improvements that work across all Persistent backends (better application-level joins)
- Better database-specific integration (better SQL joins)
- Adding more database backends

We want to stress that while most of Persistent development does occur within the Yesod (→ 4.2.8) community, there is nothing specific to Yesod about it. There is nothing stopping you from reaping its benefits in a different web framework, or on a project that has nothing to do with web development.

Further reading

http://yesodweb.com/book/persistent

6.5 Generic and Type-Level Programming

6.5.1 Unbound

Report by:	Brent Yorgey
Participants:	Stephanie Weirich, Tim Sheard
Status:	active development

Unbound is a new domain-specific language and library for working with binding structure. Implemented on top of the RepLib generic programming framework, it automatically provides operations such as alpha equivalence, capture-avoiding substitution, and free variable calculation for user-defined data types, requiring only a tiny bit of boilerplate on the part of the user. It features a simple yet rich combinator language for binding specifications, including support for pattern binding, type annotations, recursive binding, nested binding, and multiple atom types.

Work is ongoing to extend Unbound with support for generalized abstract data types (GADTs) and other features.

Further reading

- http://byorgey.wordpress.com/2011/03/28/binders-unbound/
- http://hackage.haskell.org/package/unbound
- http://code.google.com/p/replib/

6.5.2 FlexiWrap

Report by:	Iain Alexander
Status:	prototype released

A library of flexible newtype wrappers which simplify the process of selecting appropriate typeclass instances, which is particularly useful for composed types.

A proof-of-concept prototype has been released on Hackage. Work is ongoing to flesh out the typeclass instances available and provide documentation.

6.5.3 uniplate

Report by:	Neil Mitchell

See: http://www.haskell.org/communities/05-2010/html/report.html#sect5.8.1.

6.5.4 Generic Programming at Utrecht University

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

One of the research themes investigated within the Software Technology Center in the Department of Information and Computing Sciences at Utrecht University is generic programming. Over the last 10 years, we have played a central role in the development of generic programming techniques, languages, and libraries.

Currently we maintain a number of generic programming libraries and applications. We report most of them in this entry; `emgm` was reported on before (http://haskell.org/communities/05-2009/html/report.html#sect5.9.3), and our generic deriving mechanism has its own entry (→ 5.5.1).

`instant-generics` Using type families and type classes in a way similar to `multirec` and `regular`, `instant-generics` is yet another approach to generic programming, supporting a large variety of datatypes and allowing the definition of type-indexed datatypes. It was first described by Chakravarty et al., and forms the basis of one of our rewriting libaries.

`multirec` This library represents datatypes uniformly and grants access to sums (the choice between constructors), products (the sequence of constructor arguments), and recursive positions. Families of mutually recursive datatypes are supported. Functions such as *map*, *fold*, *show*, and equality are provided as examples within the library. Using the library functions on your own families of datatypes requires some boilerplate code in order to instantiate the framework, but is facilitated by the fact that `multirec` contains Template Haskell code that generates these instantiations automatically.

The `multirec` library can also be used for type-indexed datatypes. As a demonstration, the `zipper` library is available on Hackage. With this datatype-generic zipper, you can navigate values of several types.

We are still planning to extend the `multirec` library with support for parameterized datatypes and datatype compositions.

regular While **multirec** focuses on support for mutually recursive regular datatypes, **regular** supports only single regular datatypes. The approach used is similar to that of **multirec**, namely using type families to encode the pattern functor of the datatype to represent generically. There have been no major releases of the **regular** or **regular-extras** packages on Hackage since the last report. The current versions provide a number of typical generic functions, but also some less well-known but useful functions: deep *seq*, **QuickCheck**'s *arbitrary* and *coarbitrary*, and **binary**'s *get* and *put*.

syb Scrap Your Boilerplate (**syb**) has been supported by GHC since the 6.0 release. This 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.

Since **syb** has been separated from the **base** package, it can now be updated independently of GHC. We have recently released version 0.3 on Hackage, which has some minor extensions and fixes.

Annotations We presented two applications of generic annotations at the Workshop on Generic Programming 2010: selections and storage. In the former we use annotations at every recursive position of a datatype to allow for inserting position information automatically. This allows for informative parsing error messages without the need for explicitly changing the datatype to contain position information. In the latter we use the annotations as pointers to locations in the heap, allowing for transparent and efficient data structure persistency on disk.

Rewriting We also maintain two libraries for generic rewriting: a simple, earlier library based on **regular**, and the guarded rewriting library, based on **instant-generics**. The former allows for rewriting only on regular datatypes, while the latter supports more datatypes and also rewriting rules with preconditions.

We also continue to look at benchmarking and improving the performance of different libraries for generic programming (→ 5.4.2).

Further reading

http://www.cs.uu.nl/wiki/GenericProgramming

6.6 User Interfaces

6.6.1 Gtk2Hs

Report by:	Axel Simon
Participants:	Andy Stewart and 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. Gtk is the toolkit used by Gnome, one of the two major GUI toolkits on Linux. On Mac OS programs written using Gtk2Hs are run by Apple's X11 server but may also be linked against a native Aqua implementation of Gtk.

Gtk2Hs features:

- Automatic memory management (unlike some other C/C++ GUI libraries, Gtk+ provides proper support for garbage-collected languages)
- Unicode support
- High quality vector graphics using Cairo
- Extensive reference documentation
- An implementation of the "Haskell School of Expression" graphics API
- Bindings to many other libraries that build on Gtk: gio, GConf, GtkSourceView 2.0, glade, gstreamer, vte, webkit

Gtk2Hs has seen little activity and a perceived negative publicity since its web-site disappeared with the relocation of the Haskell server. As time permits, we are trying to get the web-site up and running again and recover access to our repositories. Pending this, we will apply all the good patches people sent in and do a release that works out-of-the box with GHC 7.

Further reading

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

6.6.2 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.

Haskeline supports Unicode and runs both on the native Windows console and on POSIX-compatible systems. It has a rich, user-customizable line-editing interface. Recent improvements include support for languages with wide characters and several optimizations for speed and responsiveness. Additionally, the API now provides hidden password entry and allows more control over the choice between terminal-style and file-style interactions.

Further reading

- http://trac.haskell.org/haskeline
- http://hackage.haskell.org/package/haskeline

6.6.3 CmdArgs

Report by:	Neil Mitchell
Status:	released

See: http://www.haskell.org/communities/05-2010/html/report.html#sect5.9.2.

6.7 Graphics

6.7.1 Assimp

Report by:	Joel Burget
Status:	actively developed

Assimp is a set of bindings to the Assimp Open Asset Import Library. This library can import many different types of 3D models for use in graphics. The full list of formats is available at the project website (linked below) and at the git repo for the project. Assimp is being developed alongside the Cologne ray tracer (→ 7.4.5) but could be useful in any 3D graphics project.

Further reading

- https://github.com/joelburget/assimp
- http://assimp.sourceforge.net

6.7.2 plot/plot-gtk

Report by:	Vivian McPhail
Status:	Active Development

`plot` is an embedded domain-specific language for the generation of figures. `plot-gtk` is a driver that provides a GTK widget to display figures and also a wrapper that allows interactive plotting sessions with GHCi. The package generates instructions for the Cairo renderer, which can be used to output figures in PS, PDF, PNG, and SVG file formats.

The motivation for this package is to provide a tool both for publication quality graphics and for the interactive visualisation of mathematical objects as a Haskell replacement for octave/gnuplot, matlab, and other non-Haskell numerical tools.

Users can plot functions of type `Double -> Double` and data series of type `Vector Double`, which are compatible with the high-performance `vector` package when `hmatrix` (→ 6.3.4) is installed with the `-fvector` flag.

Features:

- simple monadic interface to configure each figure and elements
- title/subtitle
- an array of plots in each figure with optional headers
- configurable axes and ticks
- configurable ranges
- linear, log, semilog ranges
- plot vectors or `(Double -> Double)` functions
- line/point/linepoint/impulse/step/area plots
- bar/histogram plots
- optional error bars
- mix and match data series types and formatting
- fully configurable text elements
- greyscale matrix visualisation

The `plot/plot-gtk` packages have just had their initial release and are available from Hackage.

Work is being done to:

- improve tick labelling and formatting
- reduce burden on the user for bar chart layout
- give elements layout tags for improved customisation
- extend the Simple interface

  3D plots are planned for a future release.

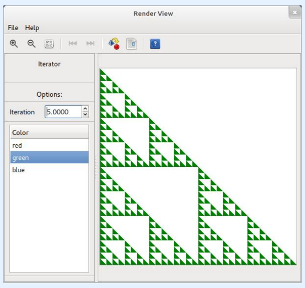
Further reading

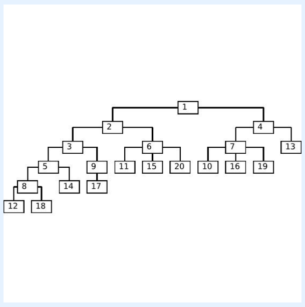
- http://hackage.haskell.org/package/plot
- http://hackage.haskell.org/package/plot-gtk

6.7.3 Craftwerk

Report by:	Malte Harder
Participants:	Jannis Harder
Status:	active development

Craftwerk is a 2D vector graphic library. The motivation was to have a graphic library that is able to generate output which can be embedded into LaTeX as well as support for rendering with Cairo. Thus the library separates the graphic's data structure from any context dependency and the aim is to support various drivers. Currently a driver for output with the TikZ package (http://sourceforge.net/projects/pgf/) for LaTeX is available. Using the additional `craftwerk-cairo` and `craftwerk-gtk` packages, direct rendering into PDF files or GTK widgets is possible. The `craftwerk-gtk` package also provides functions to generate simple user interfaces for interactive graphics.





Above, two examples are shown. In the first, you can see a screenshot of the GTK interface for interactive graphics showing a Sierpiński triangle, and the second is a simple example of a tree rendered with the Cairo driver. Graphics or figures can be created in a hierarchical fashion including the application of styles and decorations to subnodes. The current functionality includes almost the complete Cairo function set extended by arrow tips and a few primitives. The same function set is supported for TikZ output, and graphics generated with the two drivers match closely. Immediate development tasks are:

- Improvement of rendering speed in the Cairo driver.
- Better and unified text rendering capabilities.
- Refactoring of the UI module towards better usability.

Besides additional functionality, a long term goal is to support other drivers like Wumpus, Haha (ASCII rendering) or OpenGL. Craftwerk could also serve as an intermediate layer for libraries like `plot` or `chart` to enable LaTeX export. At the moment the library is still at a preliminary stage and the next step is a consolidation of a basic feature set. Any contributions or ideas are welcome and the latest code as well as experiments with other drivers are available on GitHub.

Further reading

- http://hackage.haskell.org/package/craftwerk-0.1
- http://mahrz.github.com/craftwerk

6.7.4 LambdaCube

Report by:	Csaba Hruska
Status:	experimental, active development

LambdaCube is a 3D rendering engine entirely written in Haskell.

The main goal of this project is to provide a modern and feature rich graphical backend for various Haskell projects, and in the long run it is intended to be a practical solution even for serious purposes. The engine uses Ogre3D's (http://www.ogre3d.org) mesh and material file format, therefore it should be easy to find or create new content for it. The code sits between the low-level C API (raw OpenGL, DirectX or anything equivalent; the engine core is graphics backend agnostic) and the application, and gives the user a high-level API to work with.

The most important features are the following:

- loading and displaying Ogre3D models
- resource management
- modular architecture

If your system has OpenGL and GLUT installed, the `lambdacube-examples` package should work out of the box. The engine is also integrated with the Bullet physics engine (→ 7.8.5), and you can find a running example in the `lambdacube-bullet` package.

Since the last update, there was another wave of refactoring and expansion. The most significant developments are the following:

- removed dependency on the high-level OpenGL bindings: from now on, the library only builds on `OpenGLRaw`, and the OpenGL specific code is limited to the GL render system module, which we plan to move into a separate package;
- switched to the `vect` library (from `Vec`), which was created for 3D applications from the get-go;
- simplified support for procedurally created content through vector vertex buffers, which subsumes user-supplied loaders for any format as a special case;
- introduced the LCM (LambdaCube Monad) abstraction, which hides the management of the world state and generally simplifies the engine code;
- more efficient scene management with frustum culling;
- finally added support for light sources.

Another major step forward is the creation of a complex example, which serves multiple purposes: it provides a test bed for the engine and the physics bindings, and it is also aimed to be a well-documented tutorial for future users of the library. The example will be an enhanced remake of the old racing game Stunts.

Finally, it is worth mentioning that we expect to get farther from the Ogre3D roots in the future, as the engine is evolving into a more compact and extensible architecture.

Everyone is invited to contribute! You can help the project by playing around with the code, thinking about API design, finding bugs (well, there are a lot of them anyway), creating more content to display, and generally stress testing the library as much as possible by using it in your own projects.

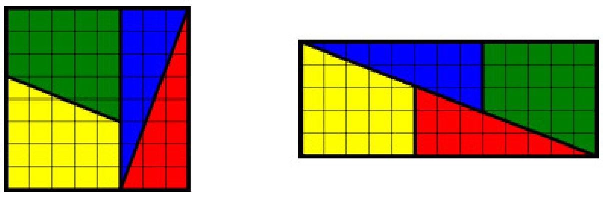
Further reading

- http://www.haskell.org/haskellwiki/LambdaCubeEngine
- http://en.wikipedia.org/wiki/Stunts_(video_game)

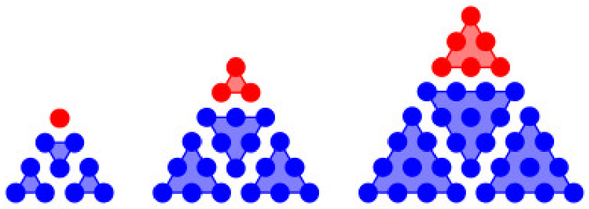
6.7.5 diagrams

Report by:	Brent Yorgey
Participants:	Ryan Yates
Status:	alpha, active development

The diagrams library provides an embedded domain-specific language for declarative drawing. The overall vision is for diagrams to become a viable alternative to DSLs like MetaPost or Asymptote, but with the advantages of being *declarative*, describing what to draw, not how to draw it, and *embedded*, putting the entire power of Haskell (and Hackage) at the service of diagram creation.



Since the last HCAR, the library has undergone a complete rewrite, and now features an elegant foundational model, pluggable rendering backends, and support for arbitrary vector spaces.



Future plans

A preview release is imminent, with a full-featured release including extensive documentation and tutorials planned by the end of the summer. Now that the core library has mostly stabilized, we hope to attract many contributors to expand the standard library and build higher-level drawing modules on top of the flexible core.

Further reading

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

6.7.6 ChalkBoard

Report by:	Andy Gill
Participants:	Kevin Matlage
Status:	ongoing

ChalkBoard is a domain specific language for describing images. The language is uncompromisingly functional and encourages the use of modern functional idioms. The novel contribution of ChalkBoard is that it uses off-the-shelf graphics cards to speed up rendering of our functional description. We always intended to use ChalkBoard to animate educational videos, as well as for processing streaming videos. Since the last HCAR report, we have added a new animation language, based round a new applicative functor, `Active`. It has been called Functional Reactive Programming, without the reactive part! The paper "Every Animation Should Have a Beginning, a Middle, and an End" talks about this addition.

A release was scheduled for November 2010.

Further reading

http://www.ittc.ku.edu/csdl/fpg/Tools/ChalkBoard

6.8 Text and Markup Languages

6.8.1 HaTeX

Report by:	Daniel Díaz
Status:	active development

HATEX is a library with the purpose of providing the possibility of integrating the script of a LATEX file in a program written in Haskell. The integration takes place through the well known monadic transformer `WriterT`, which stores in its state the LATEX code. The library provides a set of functions for adding the code, and you can include your monadic computations making use of a lifting function. HATEX is really easy to use if you know LATEX already, and only a little effort is enough otherwise. The documentation will help to learn to utilize and understand this library, with the initial guide (to be found in HaskellWiki), the extended guide "*HATEX, a monadic perspective of LATEX*", or the API documentation. The latter is not completed yet, due to the large number of entities. But, if you know LATEX, you can help to solve this.

Further reading

http://ddiaz.asofilak.es/packages/HaTeX

6.8.2 Haskell XML Toolbox

Report by:	Uwe Schmidt
Status:	seventh major release (current release: 9.1)

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 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 is partitioned into a collection of smaller packages: The core package is `hxt`. It contains a validating XML parser, an HTML parser, filters for manipulating XML/HTML and so called XML pickler for converting XML to and from native Haskell data.

Basic functionality for character handling and decoding is separated into the packages `hxt-charproperties` and `hxt-unicode`. These packages may be generally useful even for non XML projects.

HTTP access can be done with the help of the packages `hxt-http` for native Haskell HTTP access and `hxt-curl` via a libcurl binding. An alternative lazy non validating parser for XML and HTML can be found in `hxt-tagsoup`.

The XPath interpreter is in package `hxt-xpath`, the XSLT part in `hxt-xslt` and the Relax NG validator in `hxt-relaxng`. For checking the XML Schema Datatype definitions, also used with Relax NG, there is a separate and generally useful regex package `hxt-regex-xmlschema`.

The old HXT approach working with filter `hxt-filter` is still available, but currently only with hxt-8. It has not (yet) been updated to the hxt-9 mayor version.

Features

- Validating XML parser
- Very liberal HTML parser
- Lightweight lazy parser for XML/HTML based on Tagsoup (→ 6.8.3)
- Binding to the expat parser via hexpat package
- 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 and via Haskell HTTP package
- Tested with W3C XML validation suite
- Example programs
- Relax NG schema validator
- Lightweight regex library with full support of Unicode and XML Schema Datatype regular expression syntax
- An HXT Cookbook for using the toolbox and the arrow interface
- Basic XSLT support
- GitHub repository with current development versions of all packages http://github.com/UweSchmidt/hxt

Current Work

Besides maintenance work, there were some activities for better IO and parser performance. The native XML as well as the HTML parser have been optimized for speed and space. The input and output routines now work with bytestrings instead of native Haskell IO. Furthermore the XPath component has internally been changed for better performance, especially for the handling of XPath node sets.

There are some plans to further develop the Relax NG validator for full XML Schema Datatype support and for the native Relax NG schema notation. Another topic in this field is the (semi-)automatic Haskell datatype derivation out of Relax NG schemas and the generation of picklers between the schema and the Haskell types.

Further reading

The Haskell XML Toolbox Web page (http://www.fh-wedel.de/~si/HXmlToolbox/index.html) includes links to downloads, documentation, and further information.

A getting started tutorial about HXT is available in the Haskell Wiki (http://www.haskell.org/haskellwiki/HXT). The conversion between XML and native Haskell data types is described in another Wiki page (http://www.haskell.org/haskellwiki/HXT/Conversion_of_Haskell_data_from/to_XML).

6.8.3 tagsoup

Report by:	Neil Mitchell

See: http://www.haskell.org/communities/05-2010/html/report.html#sect5.11.3.

7 Applications and Projects

7.1 Education

7.1.1 Holmes, Plagiarism Detection for Haskell

Report by:	Jurriaan Hage
Participants:	Brian Vermeer, Gerben Verburg

Holmes is a tool for detecting plagiarism in Haskell programs. A prototype implementation was made by Brian Vermeer under supervision of Jurriaan Hage, in order to determine which heuristics work well. This implementation could deal only with Helium (http://www.haskell.org/communities/05-2009/html/report.html#sect2.3) programs. We found that a token stream based comparison and Moss style fingerprinting work well enough, *if* you remove template code and dead code before the comparison. Since we compute the control flow graphs anyway, we decided to also keep some form of similarity checking of control-flow graphs (particularly, to be able to deal with certain refactorings).

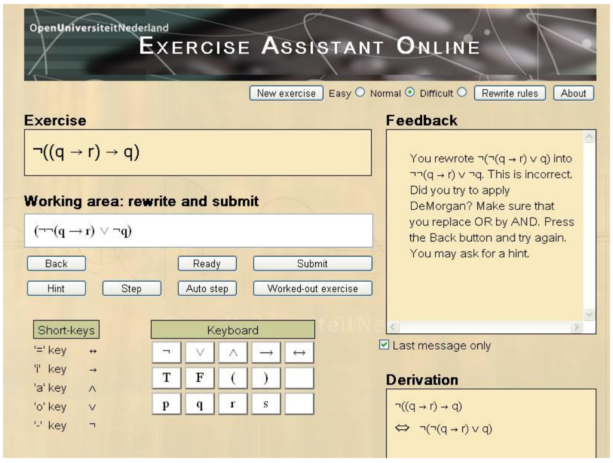
In November, Gerben Verburg started to reimplement Holmes keeping only the heuristics we figured were useful, basing that implementation on `haskell-src-exts`. I am now busy evaluating the tool on a large collection of Haskell programs, on which I aim to report at the Haskell Symposium. The tool will *not* be made available through Hackage, but will be available to lecturers on request.

We do have another control-flow graph based heuristics that seems to perform quite well in this case, and, as a sideline, we have a student in our department who has developed an algorithm for near graph-isomorphism that seems to work really well in comparing control-flow graphs in an inexact fashion.

7.1.2 Interactive Domain Reasoners (previously: Exercise Assistants)

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

The Ideas project (at Open Universiteit Nederland and Universiteit Utrecht) aims at developing interactive domain reasoners on various topics. These reasoners assist students in solving exercises incrementally by checking intermediate steps, providing feedback on how to continue, and detecting common mistakes. The reasoners are based on a strategy language, from which all feedback is derived automatically. The calculation of feedback is offered as a set of web services, enabling external (mathematical) learning environments to use our work. We currently have a binding with the Digital Mathematics Environment (DWO) of the Freudenthal Institute, the ActiveMath learning system (DFKI and Saarland University), and our own online exercise assistant that supports rewriting logical expressions into disjunctive normal form.



Exercise

$\neg((q \to r) \to q)$

Working area: rewrite and submit

$(\neg\neg(q \to r) \vee \neg q)$

Feedback

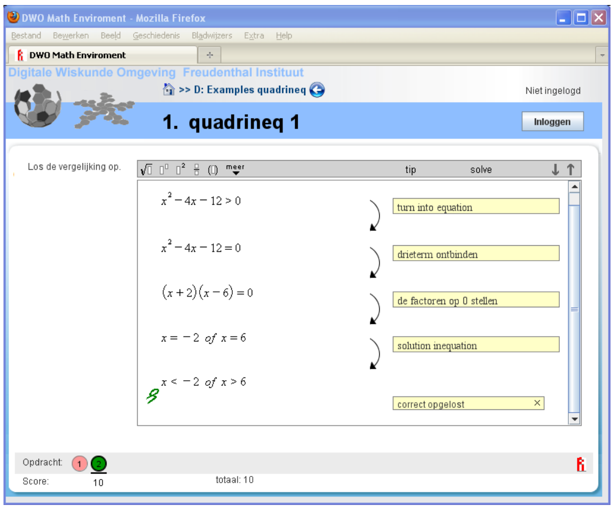
You rewrote $\neg(\neg(q \to r) \vee q)$ into $\neg\neg(q \to r) \vee \neg q$. This is incorrect. Did you try to apply DeMorgan? Make sure that you replace OR by AND. Press the Back button and try again. You may ask for a hint.

'=' key	$\leftrightarrow$
'i' key	$\to$
'a' key	$\wedge$
'o' key	$\vee$
'-' key	$\neg$

Derivation

$\neg((q \to r) \to q)$

$\Leftrightarrow \neg(\neg(q \to r) \vee q)$



1. quadrineq 1

Los de vergelijking op.

$x^2 - 4x - 12 > 0$ — turn into equation

$x^2 - 4x - 12 = 0$ — drieterm ontbinden

$(x+2)(x-6) = 0$ — de factoren op 0 stellen

$x = -2$ *of* $x = 6$ — solution inequation

$x < -2$ *of* $x > 6$ — correct opgelost

Opdracht: 1 2

Score: 10 totaal: 10

We are adding support for more exercise types, mainly at the level of high school mathematics. For example, our tool now covers simplifying expressions with exponents, rational equations, and derivatives. We have investigated how users can adapt mathematical domain reasoners to their own needs, such as the level of expertise. Recently, we have focused on designing a functional programming tutor. This tool lets you practice introductory functional programming exercises. We are also formalizing our strategy specification language, and the services that are derived from this language. This is ongoing research.

The feedback services are available as a Cabal source package.

Further reading

- Online exercise assistant (for logic), accessible from our project page.
- Bastiaan Heeren, Johan Jeuring and Alex Gerdes. Specifying Rewrite Strategies for Interactive Exercises. Mathematics in Computer Science, 3(3):349–370, 2010.
- Bastiaan Heeren, Johan Jeuring. Adapting Mathematical Domain Reasoners. International Conference on Mathematical Knowledge Management (MKM 2010).

7.1.3 Yahc

Report by:	Miguel Pagano
Participants:	Renato Cherini
Status:	testing, maintained

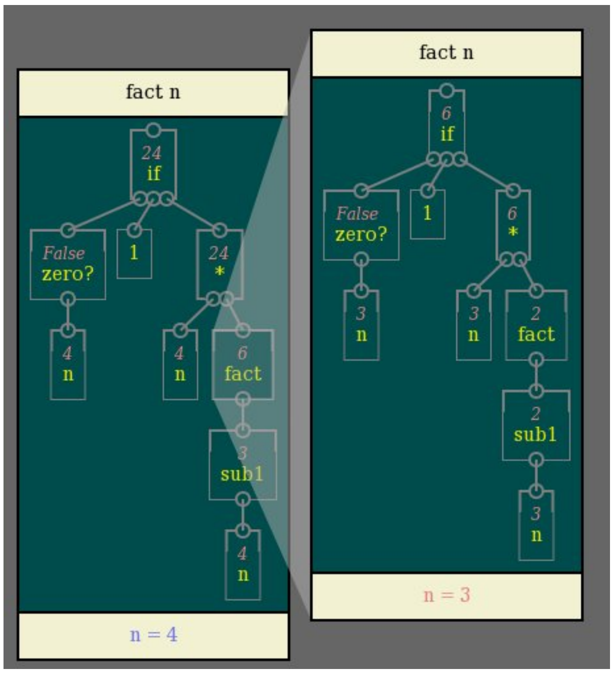
See: http://www.haskell.org/communities/05-2010/html/report.html#sect6.2.3.

7.1.4 Sifflet

Report by:	Gregory D. Weber
Status:	experimental, actively developed

Sifflet is a visual, functional programming language. Sifflet programmers define functions by drawing diagrams. Sifflet shows how a function call is evaluated on the diagram. It is intended as an aid for learning about recursion.

Here is Sifflet showing the first two levels of evaluating 4!:



Features

- Visual editor.
- Visual tracer/debugger which shows how function calls are evaluated, supporting an active learning process: Sifflet does not overwhelm students with a huge trace of function calls; it provides only as much expansion as the student requests.
- Extensive tutorial with 6,940 words and 31 pictures.
- Number, string, and list data types.
- A function "palette" with a small number of primitive functions.
- Runnable examples of compound functions.
- *New feature* (version 1.0, August 24, 2010): exports Haskell, Python 3, and Scheme code.
- *New feature* (version 1.2, October 29, 2010): Sifflet no longer indirectly depends on curl, which may make it easier for Windows users to install.

Availability

Sifflet made its public debut in May, 2010. It is available from Hackage: http://hackage.haskell.org/package/sifflet For Arch Linux, an AUR package is also available: http://aur.archlinux.org/packages.php?ID=39876

Future plans

In future releases, I hope to add these features:
- Type inference, and type declarations for exported Haskell code.
- Higher-order functions.
- Tree data and/or user-defined data types.

Further reading

- http://mypage.iu.edu/~gdweber/software/sifflet/home.html
- http://mypage.iu.edu/~gdweber/software/sifflet/doc/tutorial.html

7.2 Data Management and Visualization

7.2.1 HaskellDB

Report by:	Justin Bailey
Status:	active development

Scrap your SQL strings! The HaskellDB library provides a set of combinators based on the "relational algebra" for expressing queries, inserts, and updates. It lets you abstract over every part of your query, from conditions, to tables, to the columns returned. HaskellDB uses the HDBC family of database drivers to talk to a wide variety of databases.

Further reading

http://trac.haskell.org/haskelldb

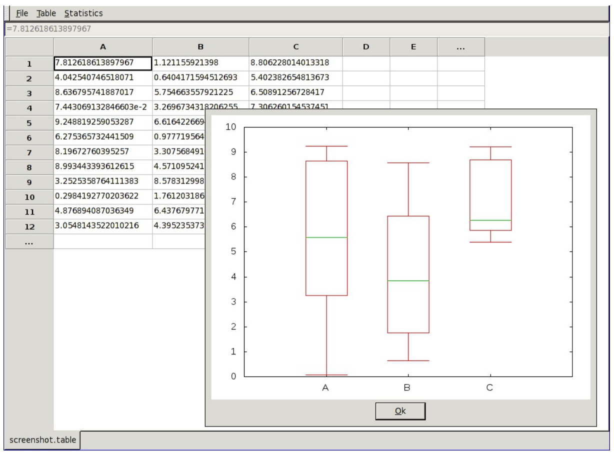
7.2.2 lhae

Report by:	Alexander Bau
Status:	in development, but stable

lhae is a simple spreadsheet application. It helps to manage your data in two-dimensional grids. Each cell in the grid contains a formula representing the stored information. Therefor lhae features a simple formula language: it supports various kinds of cell references, function calls, and conditional expressions. In order to provide automatic cell recalculation lhae keeps track of all cell dependencies.

lhae offers some table management operations like adding, deleting, inserting, transposing, and filtering of rows and columns. There are also some basic statistical methods like calculating frequency distributions, descriptive statistics, and pivot tables.

To integrate lhae in your flow of work you can import csv (character seperated values) files and export diagrams (using gnuplot).



=7.812618613897967

	A	B	C	D	E	...
1	7.812618613897967	1.121155921398	8.806228014013318			
2	4.042540746518071	0.6404171594512693	5.402382654813673			
3	8.636795741887017	5.754663557921225	6.50891256728417			
4	7.443069132846603e-2	3.269673431820625	7.30626015453745			
5	9.248819259053287	6.616422669				
6	6.275365732441509	0.977719564				
7	8.19672760395257	3.307568491				
8	8.993443393612615	4.571095241				
9	3.2525358764111383	8.578312998				
10	0.2984192770203622	1.761203186				
11	4.876894087036349	6.437679771				
12	3.0548143522010216	4.395235373				
...						

If you want to install lhae, you can use `cabal-install` by entering `cabal install lhae`.

The future development plans are mainly related to more advanced export features by supporting more of gnuplot's plotting qualities. But there also will be a more comprehensive set of functions the user can use in the formula language.

Further reading

http://www.imn.htwk-leipzig.de/~abau/lhae/

7.2.3 Pandoc

Report by:	John MacFarlane
Participants:	Andrea Rossato, Peter Wang, Paulo Tanimoto, Eric Kow, Luke Plant, Justin Bogner, Paul Rivier, Nathan Gass, Puneeth Chaganti, Josef Svenningsson, Etienne Millon, Joost Kremers
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, Textile, and reStructuredText, and it can write markdown, reStructuredText, HTML, DocBook XML, OpenDocument XML, ODT, RTF, groff man, MediaWiki markup, GNU Texinfo, LaTeX, ConTeXt, EPUB, Textile, Emacs org-mode, Slidy, and S5. Pandoc's markdown syntax includes extensions for LaTeX math, tables, definition lists, footnotes, and more.

Since the last report, many new features have been added and improvements made. Some highlights:

- Support for Textile input and output.
- Support for Emacs org-mode output.
- A new "builder" module for constructing Pandoc documents programatically.
- Support for LaTeXmath macros in markdown documents.
- Support for automatic citations and bibliographies using Andrea Rossato's citeproc-hs library.

These last two changes bring two of the most powerful features of LaTeXto pandoc.

Further reading

http://johnmacfarlane.net/pandoc/

7.2.4 Ferry (Database-Supported Program Execution)

Report by:	Torsten Grust
Participants:	George Giorgidze, Tom Schreiber, Jeroen Weijers
Status:	active development

With project *Ferry* we try to establish a connection between two somewhat distant shores: programming languages and database technology. Ferry explores how far we can push the idea of relational database engines that directly and seamlessly participate in program evaluation to support the super-fast execution of data-intensive programs written in a variety of (functional) programming languages. Relational database systems (RDBMSs) provide the best understood and most carefully engineered query processing infrastructure available today. Notwithstanding these data processing capabilities, RDBMSs are often operated as plain stores that do little more than reproduce stored data items for further processing outside the database host. With Ferry, instead, we aim to turn the database system into an efficient, capable, and highly scalable co-processor for your programming language's runtime. To this end, we search for, design, and implement new compilation strategies that map data types (e.g., nested and ordered lists, arrays, dictionaries), control structures (e.g., nested iteration, conditionals, variable assignment and reference), and idioms prevalent in functional programming and scripting languages into efficient database queries. Here, we try to push the limits of what has been considered possible (this includes algebraic data types, pattern matching, higher-order functions, and closures, to name a few).

Variants of the Ferry technology have been used

- to enhance the SQL code generator in Philip Wadler's *Links*, such that a significantly larger class of Links programs may be considered *databaseable* now, and
- to create a capable and efficient version of *LINQ to SQL* provider (plugging into the Microsoft .NET Language Integrated Query framework),
- to create a deep embedding of queries, nicknamed *Switch*, into the object-oriented language Ruby, effectively turning Ruby programs over arrays into relational queries, and
- to create an integrated query facility for Haskell, called *Database supported Haskell* (DSH).

We have re-implemented the Ferry compiler in Haskell (using GHC). The Ferry compiler is used as part of the embedded query language DSH for Haskell. DSH is suited to handle large scale data (e.g., social networks) in Haskell programs with familiar Haskell syntax. The DSH library and the FerryCore package it uses are available on Hackage (http://hackage.haskell.org/package/DSH).

Bring Back Monad Comprehensions. We are currently working on a new, monad-based, version of DSH which can be used to write queries using *Monad Comprehensions*. An effort of our research team will reintroduce support for Monad Comprehensions in the next release of GHC (version 7.2). The progress of this work is tracked at http://hackage.haskell.org/trac/ghc/ticket/4370.

Future plans

Ferry employs a compilation strategy revolving around the concept of *loop lifting* that appears to have quite close and interesting connections to the *flattening transformation* employed by Data Parallel Haskell. Indeed, Ferry understands the relational query engine as being a specific kind of data-parallel machine. The exact connection between Ferry and Data Parallel Haskell remains to be explored.

Further reading

http://www.ferry-lang.org

7.2.5 The Proxima 2.0 Generic Editor

Report by:	Martijn Schrage
Participants:	Lambert Meertens, Doaitse Swierstra
Status:	actively developed

See: http://www.haskell.org/communities/05-2010/html/report.html#sect6.4.5.

7.3 Functional Reactive Programming

7.3.1 reactive-banana

Report by:	Heinrich Apfelmus
Status:	active development

Reactive-banana is a small library for functional reactive programming (FRP).

The goal is to create a solid foundation for anything FRP-related.

- Users can finally start experimenting with graphical user interfaces based on FRP since the library can be hooked into *any* existing event-based framework and comes with ample documentation.
- FRP implementors will have a reference for a simple semantics with a working implementation.
- No more spooky time leaks and efficiency concerns. Predicting space & time usage should be straightforward.

Version 0.2 of the reactive-banana library has been released on Hackage. It provides a solid push-based implementation of a subset of the semantics for FRP pioneered by Conal Elliott.

Current development focuses on providing tutorials, documentation and examples for the library. Furthermore, the author is writing an example application to test and refine the FRP approach to GUI programming.

Further reading

- Cabal package and link to source code: http://hackage.haskell.org/package/reactive
- Developer blog: http://apfelmus.nfshost.com/blog.html

7.3.2 Functional Hybrid Modelling

Report by:	George Giorgidze
Participants:	Joey Capper, Henrik Nilsson
Status:	active research and development

The goal of the FHM project is to gain a better foundational understanding of non-causal, hybrid modelling and simulation languages for physical systems and ultimately to improve on their capabilities. At present, our central research vehicle to this end is the design and implementation of a new such language centred around a small set of core notions that capture the essence of the domain.

Causal modelling languages are closely related to synchronous data-flow languages. They model system behaviour using ordinary differential equations (ODEs) in explicit form. That is, cause-effect relationship between variables must be explicitly specified by the modeller. In contrast, non-causal languages model system behaviour 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 modelling 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 modelling which combines functional programming and non-causal modelling. In particular, the FHM approach proposes modelling with first class models (defined by continuous DAEs) using combinators for their composition and discrete switching. The discrete switching combinators enable modelling of hybrid systems (i.e., systems that exhibit both continuous and discrete dynamic behaviour). 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 and enables modellers to use the domain specific syntax in their models. The present prototype implementation of Hydra enables modelling with first class models and supports combinators for their composition and discrete switching.

We implemented support for dynamic switching among models that are computed at the point when they are being "switched in". Models that are computed at run-time are just-in-time (JIT) compiled to efficient machine code. This allows efficient simulation of highly structurally dynamic systems (i.e., systems where the number of structural configurations is large, unbounded or impossible to determine in advance). This goes beyond to what current state-of-the-art non-causal modelling languages can model. The implementation techniques that we developed should benefit other modelling and simulation languages as well.

We are also exploring ways of utilising the type system to provide stronger correctness guarantees and to provide more compile time reassurances that our system of equations is not unsolvable. Properties such as equational balance (ensuring that the number of equations and unknowns are balance) and ensuring the solvability of locally scoped variables are among our goals. Dependent types have been adopted as the tool for expressing these static guarantees. However, we believe that more practical type systems (such as system F) could be conservatively extended to make FHM safer without compromising their usability.

Recently, in an effort to showcase FHM and Hydra to the wider modelling and simulation community, we have modelled and simulated a number of challenging physical systems that current non-causal modelling languages can not handle (see the papers linked below).

Further reading

The implementation of Hydra and related papers are available from http://www.cs.nott.ac.uk/~ggg/.

7.3.3 Elerea

Report by:	Patai Gergely
Status:	experimental, active

Elerea (Eventless reactivity) is a tiny discrete time FRP implementation without the notion of event-based switching and sampling, with first-class signals (time-varying values). Reactivity is provided through various higher-order constructs that also allow the user to work with arbitrary time-varying structures containing live signals.

Stateful signals can be safely generated at any time through a specialised monad, while stateless combinators can be used in a purely applicative style. Elerea signals can be defined recursively, and external input is trivial to attach. The library comes in three major variants, which all have precise denotational semantics:

- `Simple`: signals are plain discrete streams isomorphic to functions over natural numbers;
- `Param`: adds a globally accessible input signal for convenience;
- `Clocked`: adds the ability to freeze whole subnetworks at will.

The code is readily available via cabal-install in the `elerea` package. You are advised to install `elerea-examples` as well to get an idea how to build non-trivial systems with it. The examples are separated in order to minimize the dependencies of the core library. The experimental branch is showcased by Dungeons of Wor, found in the `dow` package (http://www.haskell.org/communities/05-2010/html/report.html#sect6.11.2). Additionally, the basic idea behind the experimental branch is laid out in the WFLP 2010 article *Efficient and Compositional Higher-Order Streams.*

Since the last report, the `Delayed` variant of the library was deprecated, because it proved to be more trouble than worth (mostly because automatic delays break referential transparency). Also, the `Clocked` branch needs some overhaul due to the trickiness of clock semantics. This is ongoing work.

Further reading

- http://hackage.haskell.org/package/elerea
- http://hackage.haskell.org/package/elerea-examples
- http://hackage.haskell.org/package/dow
- http://sgate.emt.bme.hu/documents/patai/publications/PataiWFLP2010.pdf
- http://babel.ls.fi.upm.es/events/wflp2010/video/video-08.html (WFLP talk)

7.4 Audio and Graphics

7.4.1 Audio Signal Processing

Report by:	Henning Thielemann
Status:	experimental, active development

In this project, audio signal algorithms are written in Haskell, that is, no binding to existing sound synthesis systems like SuperCollider. The highlights are:

- based on the Numeric Prelude framework (http://haskell.org/communities/05-2009/html/report.html#sect5.6.2).
- support for physical units while maintaining efficiency,
- frameworks for abstraction from sample rate, that is, the sampling rate can be omitted in most parts of a signal processing expression.
- We checked several low-level implementations in order to achieve reasonable speed. We complement the standard list type with a lazy `StorableVector` structure and a `StateT s Maybe a` generator, like in stream-fusion. Now, both our custom signal generator type and the Stream type from stream-fusion can be fused to work directly on storable vectors.
- 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 for sharing on a per sample basis and in feedback loops where they statically warrant that no future data is accessed.
- Type class framework for unifying lazy time values and signals expressed by lists, storable vectors or signal generators.
- Connection to ALSA bindings, in order to provide real-time sound synthesis controlled by MIDI events from keyboards or sequencers.
- A real-time software synthesizer that employs Just-In-Time-compilation and vector instructions provided by the Low-Level Virtual Machine (http://llvm.org/)

Recent advances are:

- Implementation of the Fast Fourier Transform for all signal lengths, that can be used both for complex numbers and integer residue class fields.
- Novel algorithm for white noise generation that adapts to the sampling rate.

Further reading

- http://www.haskell.org/haskellwiki/Synthesizer
- http://arxiv.org/abs/1103.4118

7.4.2 Tidal, Texture and Live Music with Haskell

Report by:	Alex McLean
Status:	experimental

For a number of years, I have been improvising live music with Haskell. I have made a pattern library called Tidal and have most recently been working on an experimental visual language on front of that called Texture (formerly known as Text, and I am still in the process of renaming it). There are various videos and some more information on my homepage.

I have been using Tidal and its predecessors in live performance for some years, as shown this video of a performance in Norway: http://piksel.blip.tv/file/4521577/. The quality of the recording is not perfect, but it does show people dancing to Haskell. This performance was with Dave Griffiths (who used his own visual Scheme language SchemeBricks), we perform together (usually as a trio with Adrian Ward) as Slub, and are available for bookings.

Texture is rather experimental, but I recently ran a workshop with it, and got six non-programmers writing Haskell code to improvised music of the acid techno genre together over a few hours.

The code is available at http://darcs.slab.org/, but is undocumented and difficult to get running. Those interested in dabbling in this area would probably be better off looking at hsc3 or haskore. Conductive is another new and interesting library.

At the moment I am finishing off my PhD thesis on a related topic, after that I intend to spend some time packaging Tidal and Texture properly.

Folks interested in Haskell and music, as well as other artforms should consider signing up to the haskell art mailing list.

Further reading

http://yaxu.org/

7.4.3 Hemkay

Report by:	Patai Gergely
Status:	experimental, active

Hemkay (An M.K. Player Whose Name Starts with an H) is a simple music module player that performs all the mixing in Haskell. It supports the popular ProTracker format and some of its variations with different numbers of channels. The device independent mixing functionality can be found in the `hemkay-core` package.

The current version of the player uses the list-based PortAudio bindings for playback, which is highly inefficient.

Since the last update, the mixer went through some performance optimisations. However, the improved mixing performance can only be exploited either through the alternative callback interface of PortAudio (check the `hemkay/callback` branch on GitHub), or through the OpenAL version (`hemkay/openal` branch). Out of the two, the PortAudio version is significantly more efficient, but it is prone to random crashes. Note that this alternative PortAudio binding is only available on GitHub.

Further reading

- http://hackage.haskell.org/package/hemkay-core
- http://hackage.haskell.org/package/hemkay
- http://en.wikipedia.org/wiki/MOD_(file_format)
- https://github.com/cobbpg/hemkay
- https://github.com/mietek/portaudio

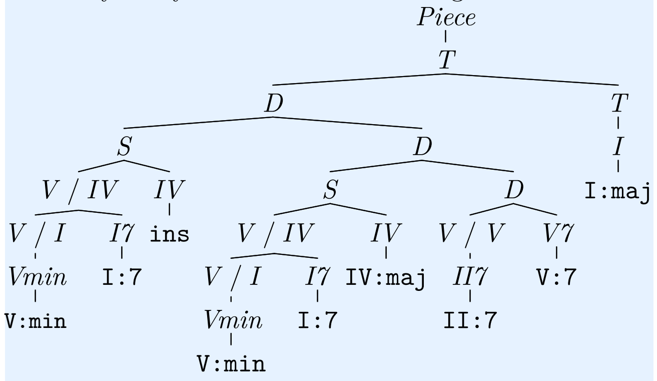
7.4.4 Functional Modelling of Musical Harmony

Report by:	José Pedro Magalhães
Participants:	W. Bas de Haas
Status:	actively developed

Music theory has been essential in composing and performing music for centuries. Within Western tonal music, from the early Baroque on to modern-day jazz and pop music, the function of chords within a chord sequence can be explained by harmony theory. Although Western tonal harmony theory is a thoroughly studied area, formalising this theory is a hard problem.

We have developed a formalisation of the rules of tonal harmony as a Haskell (generalized) algebraic datatype. Given a sequence of chord labels, the harmonic function of a chord in its tonal context is automatically derived. For this, we use several advanced functional programming techniques, such as type-level computations, datatype-generic programming, and error-correcting parsers. Our functional model of harmony offers various benefits: it can be used to define harmonic similarity measures and facilitate music retrieval, or it can help musicologists in batch-analysing large corpora of digitised scores, for instance. We have a draft version of a report detailing this project.

As an example, we show a tree representation of the harmony analysis of a short music fragment:



This tree is a visual representation of a value of a Haskell datatype encoding musical harmony, with common notions such as tonic, dominant, etc. Such trees are generated from input sequences of chord labels such as `C:maj F:maj G:7 C:Maj`.

We hope to release our code on Hackage soon.

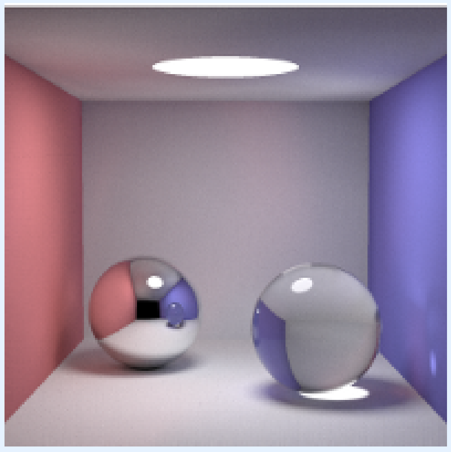
Further reading

http://dreixel.net/research/pdf/fmmh_draft.pdf

7.4.5 Cologne

Report by:	Joel Burget
Status:	actively developed

Cologne is a ray tracer being developed in Haskell. The goal is to produce a fun and relatively performant ray tracer. The project has been slowed down recently as my main focus has been on importing more complex models through the Assimp project (→ 6.7.1), but development should pick up this summer. Check out this render of the smallpt scene:



Further reading

https://github.com/joelburget/Cologne

7.4.6 easyVision

Report by:	Alberto Ruiz
Status:	experimental, active development

The *easyVision* project is a collection of experimental libraries for computer vision and image processing. The low level computations are internally implemented by optimized libraries (IPP, HOpenGL, hmatrix (→ 6.3.4), etc.). Once appropriate geometric primitives have been extracted by the image processing wrappers we can define interesting computations using high level combinators.

Further reading

http://perception.inf.um.es/easyVision

7.5 Hardware Design

7.5.1 CλaSH

Report by:	Christiaan Baaij
Participants:	Arjan Boeijink, Jan Kuper, Anja Niedermeier, Matthijs Kooijman, Marco Gerards
Status:	experimental

CλaSH (CAES Language for Synchronous Hardware) is a functional hardware description language that borrows both its syntax and semantics from Haskell. The clock is implicit for the descriptions made in CλaSH: the behaviour of the circuit is described as transition from the current state to the next, which occurs every clock cycle. The current state is an input of such a transition function, and the updated state part of its result tuple. As descriptions are also valid Haskell, simulations can simply be performed by a Haskell compiler/interpreter (GHC only, due to the use of type families).

Instead of being an embedded language such as ForSyDe (→ 7.5.2) and Lava (→ 2.6)(→ 7.5.3)(→ 9.9), CλaSH has a compiler which can translate Haskell to synthesizable VHDL. The compiler has support for, amongst others: polymorphism, higher-order functions, user-defined algebraic datatypes, and all of Haskell's choice mechanisms. The CλaSH compiler uses GHC for parsing, de-sugaring, and type-checking. The resulting Core-language description is then transformed into a normal form, from which a translation to VHDL is direct. The transformation system uses a set of rewrite rules which are exhaustively applied until a description is in normal form. Examples of these rewrite rules are β-reduction and η-expansion, but also transformations to transform higher-order functions to first-order functions, and transformation for the specialization of polymorphic functions.

The CλaSH compiler was first presented to the community, after 7 months of work, at the Haskell 2009 symposium in Edinburgh, Scotland. Support for arrows and the corresponding syntax, which eases the composition of transition functions, was added in July 2010 and was subsequently presented at IFL 2010 in Alphen a/d Rijn, The Netherlands.

The CλaSH compiler, available as a library, can be found both on Hackage (http://hackage.haskell.org/package/clash, stable) and github (http://github.com/christiaanb/clash/, development). The compiler/interpreter is also available as an executable, which is basically the GHC binary extended with the CλaSH library, on the CλaSH website (http://clash.ewi.utwente.nl).

What is new?

There is now simulation and synthesis support for hardware descriptions that have multiple clock domains, starting with version 0.1.3.0 of CλaSH. Example usage of multiple clock domains is explained here: http://www.haskell.org/pipermail/haskell-cafe/2011-March/090471.html. The code for the demo (which uses multiple clock domains) we did at the DATE'11 conference is available here: http://github.com/christiaanb/DE1-Cyclone-II-FPGA-Board-Support-Package.

Further reading

http://clash.ewi.utwente.nl

7.5.2 ForSyDe

Report by:	Ingo Sander
Participants:	Hosein Attarzadeh, Alfonso Acosta, Axel Jantsch, Jun Zhu
Status:	experimental

See: http://www.haskell.org/communities/05-2010/html/report.html#sect6.8.1.

7.5.3 Kansas Lava

Report by:	Andy Gill
Participants:	Tristan Bull, Andrew Farmer, Ed Komp
Status:	ongoing

Kansas Lava is a modern implementation of a hardware description language that uses functions to express hardware components, and leverages the abstractions in Haskell to build complex circuits. Lava, the given name for a family of Haskell based hardware description libraries (→ 2.6)(→ 9.9), is an idiomatic way of expressing hardware in Haskell which allows for simulation and synthesis to hardware.

Though there has been no public release (yet), we have made considerable progress with Kansas Lava. We have generated several large telemetry circuits, which have been synthesized and tested on real hardware, running at speeds comparable to other implementation techniques. A talk about internals of Kansas Lava was presented by Andrew Farmer at the Haskell implementors workshop in October, and the talk and slides are available online.

Jun Inoue from Rice University visited CSDL for October and November, to help connect his "staging" work with the Kansas Lava work. A release of Kansas Lava release was planned for the end of 2010.

Further reading

http://www.ittc.ku.edu/csdl/fpg/Tools/KansasLava

7.6 Proof Assistants and Reasoning

7.6.1 Automated Termination Analyzer for Haskell

Report by:	Jürgen Giesl
Participants:	Matthias Raffelsieper, Peter Schneider-Kamp, Stephan Swiderski, René Thiemann
Status:	actively developed

There are many powerful techniques for automated termination analysis of term rewriting. However, up to now they have hardly been used for real programming languages. We developed an approach which permits the application of existing techniques from term rewriting to prove termination of most functions defined in Haskell programs. In particular, we show how termination techniques for ordinary rewriting can be used to handle those features of Haskell which are missing in term rewriting (e.g., lazy evaluation, polymorphic types, and higher-order functions). We implemented our results in the termination prover AProVE. When testing it on existing standard Haskell-libraries, it turned out that AProVE can fully automatically prove termination of the vast majority of the functions in the libraries.

Further reading

- For details on our approach:

  J. Giesl, M. Raffelsieper, P. Schneider-Kamp, S. Swiderski, and R. Thiemann. Automated Termination Proofs for Haskell by Term Rewriting. *ACM Transactions on Programming Languages and Systems*, 33(2), 2011. http://dx.doi.org/10.1145/1890028.1890030

- To access the implementation via a web interface and for further information on our experiments:

  http://aprove.informatik.rwth-aachen.de/eval/Haskell/

7.6.2 Zeno — Inductive Theorem Proving for Haskell Programs

Report by:	Will Sonnex
Participants:	Sophia Drossopoulou, Susan Eisenbach
Status:	Alpha 0.1.1

Zeno is a fully automated inductive theorem proving tool for proving properties of Haskell functions. You can express a property such as `takeWhile p xs ++ dropWhile p xs === xs` and it will prove it to be true for all values of `p :: a -> Bool` and `xs :: [a]`, over all types `a`, using only the function definitions.

After its most recent update Zeno can now reason about polymorphic types/functions, and you express the properties to be proven in Haskell itself (thanks to SPJ for the suggestion). It still cannot use all of Haskell's syntax: you cannot have internal functions (let/where can only assign values), and you cannot use type-classed polymorphic variables in function definitions — you will have to create a monomorphic instance of the function — but I hope to have these added reasonably soon. It is also still missing primitive-types/IO/imports so it still cannot be used with any real-world Haskell code, it is more a bit of theorem proving "fun".

Another feature is that Zeno lists all the sub-properties it has proven within each proof. When it verifies insertion-sort (`sorted (insertsort xs) === True`) it also proves the antisymmetry of `<=` and that the `insert` function preserves the `sorted` property.

You can try Zeno out at http://www.doc.ic.ac.uk/~ws506/tryzeno, the example code file given there has some provable properties about a few Prelude functions among other things. If you want the source code, it is available at http://code.google.com/p/zeno but I would advise you to use one of the branched versions, I make no guarantee that the trunk will even compile.

Further reading

http://www.doc.ic.ac.uk/~ws506/tryzeno

7.6.3 Free Theorems for Haskell

Report by:	Janis Voigtländer
Participants:	Daniel Seidel, Matthias Bartsch

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 journal version of our earlier “Taming Selective Strictness” paper has now appeared (http://www.iai.uni-bonn.de/~jv/acta.pdf). Also, we have been looking at the quantitative content of free theorems (http://www.iai.uni-bonn.de/~jv/qapl11.pdf).

On the practical side, we maintain a library and tools for generating free theorems from Haskell types, originally implemented by Sascha Böhme and with contributions from Joachim Breitner and now Matthias Bartsch. 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://www-ps.iai.uni-bonn.de/ft/. 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
- plain text, LaTeX source, PDF, and inline graphics output with nicely typeset theorems

Matthias is still working on refactoring the internals of the generator, opening up possibilities for better control by the user, as well as for generating new forms of free theorems.

Further reading

http://www.iai.uni-bonn.de/~jv/project/

7.6.4 Streaming Component Combinators

Report by:	Mario Blažević
Status:	experimental, actively developed

See: http://www.haskell.org/communities/11-2010/html/report.html#sect9.6.5.

7.6.5 CSP-M Animator and Model Checker

Report by:	Marc Fontaine
Status:	active development, download available

See: http://www.haskell.org/communities/05-2010/html/report.html#sect6.7.5.

7.6.6 Swish

Report by:	Douglas Burke
Participants:	Graham Klyne, Vasili I Galchin
Status:	experimental

Swish is a framework for performing deductions in RDF data using a variety of techniques. Swish is conceived as a toolkit for experimenting with RDF inference, and for implementing stand-alone RDF file processors (usable in similar style to CWM, but with a view to being extensible in declarative style through added Haskell function and data value declarations). It explores Haskell as “a scripting language for the Semantic Web”, is a work-in-progress, and currently incorporates:

- Support for both Notation3 and NTriples formats.
- RDF graph isomorphism testing and merging.
- Display of differences between RDF graphs.
- Inference operations in forward chaining, backward chaining and proof-checking modes.
- Simple Horn-style rule implementations, extendable through variable binding modifiers and filters.
- Class restriction rule implementation, primarily for datatype inferences.
- RDF formal semantics entailment rule implementation.
- Complete, ready-to-run, command-line and script-driven programs.

Current Work

The version on Hackage has recently been updated from 0.2.1 to the 0.3 series; the main changes were to make the package build with recent Haskell Platform releases, updates to match the latest N3 specification, and addition of the NTriples format. Minor bug fixes and improvements have been made to this series.

Future plans

The major planned changes are a move to using the Data.Text module, addition of an RDF/XML parser, profiling and further clean up of the code. Community input — whether it be patches, new code or just feature requests — are more than welcome.

Further reading

- https://bitbucket.org/doug_burke/swish/
- http://www.ninebynine.org/RDFNotes/Swish/Intro.html

7.7 Natural Language Processing

7.7.1 NLP

Report by:	Eric Kow

The Haskell Natural Language Processing community aims to make Haskell a more useful and more popular language for NLP. The community provides a mailing list, Wiki and hosting for source code repositories via the Haskell community server.

The Haskell NLP community was founded in March 2009. The list is still growing slowly as people grow increasingly interested in both natural language processing, and in Haskell. Since the last report, there have been a couple of new releases in the community:

- *Haskell for the Working Programmer*: Two members of the Haskell NLP community started work on a book that aims to provide an introduction to Haskell and Natural Language Processing. This work-in-progress is freely available from http://nlpwp.org/
- CLT toolkit: A set of mutually and externally compatible state-of-the-art open source language technology tools and accompanying linguistic resource. This put work by Chalmers and Gothenburg University around Haskell and NLP into a single package, including Grammatical Framework and Functional Morphology. http://www.clt.gu.se/clt-toolkit
- alpino-tools: A package for processing training data of the Alpino parse disambiguation and fluency ranking components. Alpino is a wide-coverage parser/generator for Dutch. The training data uses a simple format, which may make it useful for other systems.

At the present, the mailing list is mainly used to make announcements to the Haskell NLP community. We hope in the future that it will expand to include broader discussions on creating libraries and bindings that would be most useful to us and ways of spreading awareness about Haskell in the NLP world.

Further reading

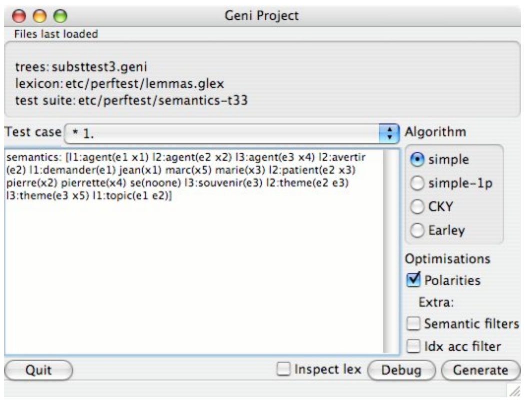
http://projects.haskell.org/nlp

7.7.2 GenI

Report by: Eric Kow

GenI is a surface realizer for Tree Adjoining Grammars. Surface realization can be seen a subtask of natural language generation (producing natural language utterances, e.g., English texts, out of abstract inputs). GenI in particular takes a Feature Based Lexicalized Tree Adjoining Grammar and an input semantics (a conjunction of first order terms), and produces the set of sentences associated with 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, with dual-licensing available for commercial purposes.

Work on GenI has begun anew. Since May 2011, Eric is working with Computational Linguistics Ltd and SRI international to develop new features for GenI and improve its scalability and performance for use in an interactive tutoring application. We are excited to see GenI potentially being used in the real world!



trees: substtest3.geni
lexicon: etc/perftest/lemmas.glex
test suite: etc/perftest/semantics-t33

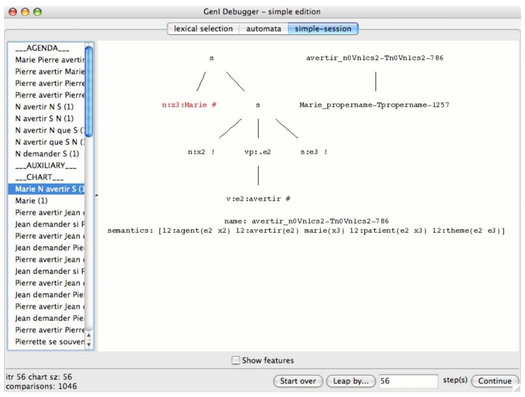
Test case: * 1.

semantics: [l1:agent(e1 x1) l2:agent(e2 x2) l3:agent(e3 x4) l2:avertir (e2) l1:demander(e1) jean(x1) marc(x5) marie(x3) l2:patient(e2 x3) pierre(x2) pierrette(x4) se(noone) l3:souvenir(e3) l2:theme(e2 e3) l3:theme(e3 x5) l1:topic(e1 e2)]

Algorithm: [✓ simple] simple-1p CKY Earley

Optimisations: [✓ Polarities]
Extra: Semantic filters, Idx acc filter

Inspect lex



```
___AGENDA___
Marie Pierre avertir
Pierre avertir Marie
Pierre avertir Pierre
Pierre avertir Pierre
N avertir N S (1)
N avertir S N (1)
N avertir N que S (
N avertir que S N (
N demander S (1)
___AUXILIARY___
___CHART___
Marie N avertir S (1
Marie (1)
Pierre avertir Jean
Jean demander si P
Pierre avertir Jean
Jean demander Pie
Pierre avertir Jean
Jean demander si P
Pierre avertir Jean
Jean demander Pie
Pierre avertir Jean
Jean demander Pie
Pierre avertir Pierre
Pierrette se souven
```

name: avertir_n0Vnlcs2-Tn0Vnlcs2-786
semantics: [l2:agent(e2 x2) l2:avertir(e2) marie(x3) l2:patient(e2 x3) l2:theme(e2 e3)]

itr 56 chart sz: 56
comparisons: 1046

56 step(s)

GenI is available on Hackage, and can be installed via cabal-install. Our most recent release of GenI was version 0.20.2 (2009-12-02), with some bugfixes and simplifications. For more information, please contact us on the geni-users mailing list.

Further reading

- http://projects.haskell.org/GenI
- Paper from Haskell Workshop 2006: http://hal.inria.fr/inria-00088787/en
- http://websympa.loria.fr/wwsympa/info/geni-users

7.7.3 Grammatical Framework

Report by: Krasimir Angelov
Participants: Olga Caprotti, Grégoire Détrez, Ramona Enache, Thomas Hallgren, Aarne Ranta

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 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 Amharic, Bulgarian, Catalan, Danish, Dutch, English, Finnish, French, German, Interlingua, Italian, Norwegian, Russian, Spanish, Swedish and Urdu. There are also incomplete grammars for Arabic, Latin, Thai, Turkish, and Hindi. On top of these grammars a user with limited linguistic background can build application grammars for a particular domain.

We are planning the release of GF 3.2 before the end of this year. The latest development around GF is mostly driven by the new research project MOLTO (http://www.molto-project.eu/) which focuses on tools for multilingual online translation. This are some of the features that can be expected in the new release:

- We improve our web front-end so the users (translators, content authors, etc.) will work in a more comfortable environment. There are some demos on the GF home page.
- There is work in progress on a new editor which combines free text authoring with structural editing.
- Now there is a web based browser which the grammarians can use to explore the content of the grammars. The functionality is similar to what you would expect from tools like Haddock for Haskell.
- The support for dependent types is becoming stable. All abstract syntax trees are type checked before they are used. We use the same type checking algorithm as in Agda and we support the Agda style of implicit arguments. If the abstract syntax in some grammar uses dependent types then the parser checks whether the parsed sentence is semantically consistent. When GF is used for parsing formal languages like C/C++ then the concrete syntax of the grammar describes the syntax of the language while the dependent types in the abstract syntax can be used to specify which programs are well-typed.
- We also support random and exhaustive generation of lambda terms of a given type. Since the type system supports dependent types, the generation is actually equivalent to proving a theorem in first-order logic. In fact, we build our own in-house theorem prover.
- Previously the parser in GF either produced some result or just failed. Now it is much more friendly and could detect where exactly is the problem. If there is a syntax error then the token position is reported. If the parsing is successful but none of the possible parse trees is semantically consistent then the inconsistent phrase is located and a detailed error message is reported. In some cases semantic inconsistencies can be detected even before the sentence is complete. In formal languages, this corresponds to type checking of incomplete programs.
- There is an alternative implementation of the GF interpreter in Java which makes it possible to run applications on platforms where Haskell is not well supported. For instance we developed a user interface which works on Android phones.

Further reading

http://www.grammaticalframework.org/

7.8 Others

7.8.1 GenProg — Genetic Programming Library

Report by:	Jan Šnajder
Status:	experimental

The GenProg library is a framework for genetic programming. Genetic programming is an evolutionary technique, inspired by biological evolution, to evolve programs for solving specific problems. A genetic program is represented as an abstract syntax tree and associated with a custom-defined fitness value indicating the quality of the solution. Starting from a randomly generated initial population of genetic programs, the genetic operators of selection, crossover, and (occasionally) mutation are used to evolve programs of increasingly better quality. Standard reference is John Koza's *Genetic programming: On the Programming of Computers by Means of Natural Selection.*

In GenProg, a genetic program is represented by a value of an algebraic datatype. To use a datatype as a genetic program, it suffices to define it as an instance of the `GenProg` typeclass. Any custom datatype can be made an instance of the `GenProg` typeclass. In particular, to use instances of the `Data` typeclass as genetic programs it suffices to define two simple functions: one for the generation of random terminal nodes and another for the generation of random nonterminal nodes. The evolution is governed by several user defined parameters, such as population size, crossover and mutation probabilities, termination criterion, and mutation function. The package is available on Hackage.

Further reading

http://hackage.haskell.org/package/genprog

7.8.2 Manatee

Report by:	Andy Stewart
Status:	active development

Manatee's aim is to build a Haskell Operating System.

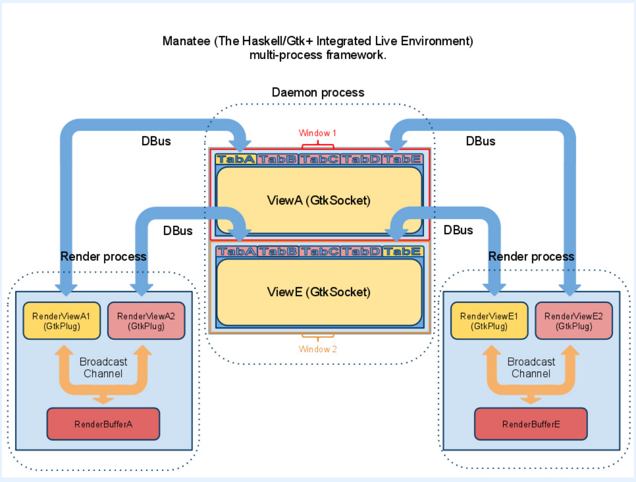
I am an Emacs fan (http://www.emacswiki.org/emacs/AndyStewart) that uses Emacs everyday for everything. But Emacs does not support multi-thread and is not safe enough. So I am building my own Haskell integrated environment — Manatee.

You can write any application in it, and the Manatee framework will mix your application with the current environment. And, most importantly, it gives you a uniform experience with different applications.

Framework

Manatee uses a multi-process framework that makes the extension and the core running in separate processes to protect the application. It will minimize your losses when some unexpected exception happens in the current application; you just need to close/reload the current tab, any other application and the core are still running safely.

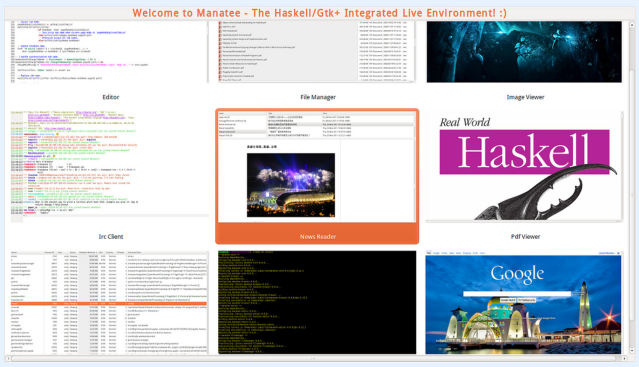
Manatee uses a Model-View split design; you can split the current window to get different views for the same buffer (a bit like Emacs's buffers and windows). Then you can mix any applications together with this design for working efficiently.



Future plans

I have written the below applications in Manatee:
- Web Browser
- Download Manager
- Editor
- File Manager
- Image Viewer
- IRC Client
- Multimedia Player
- PDF Viewer
- Process Manager
- News Reader
- Terminal

More applications are in development, you are welcome to join us!



Further reading

- Screenshots: http://goo.gl/MkVw
- Videos: http://www.youtube.com/watch?v=weS6zys3U8k, http://www.youtube.com/watch?v=A3DgKDVkyeM
- Wiki page: http://haskell.org/haskellwiki/Manatee

Contact

- Mailing lists: ⟨manatee-user@googlegroups.com⟩, ⟨manatee-develop@googlegroups.com⟩
- IRC channel: irc.freenode.net, 6667, ##manatee

7.8.3 xmonad

Report by:	Gwern Branwen
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.

Development since the last report has continued apace, with versions 0.8, 0.8.1, 0.9 and 0.9.1 released, with simultaneous releases of the XMonadContrib library of customizations and extensions, which has now grown to no less than 205 modules encompassing a dizzying array of features.

Details of changes between releases can be found in the release notes:
- http://haskell.org/haskellwiki/Xmonad/Notable_changes_since_0.7
- http://haskell.org/haskellwiki/Xmonad/Notable_changes_since_0.8
- http://haskell.org/haskellwiki/Xmonad/Notable_changes_since_0.9
- XMonad.Config.PlainConfig allows writing configs in a more 'normal' style, and not raw Haskell

- Supports using local modules in xmonad.hs; for example: to use definitions from ~/.xmonad/lib/XMonad/Stack/MyAdditions.hs
- xmonad –restart CLI option
- xmonad –replace CLI option
- XMonad.Prompt now has customizable keymaps
- Actions.GridSelect - a GUI menu for selecting windows or workspaces & substring search on window names
- Actions.OnScreen
- Extensions now can have state
- Actions.SpawnOn - uses state to spawn applications on the workspace the user was originally on, and not where the user happens to be
- Markdown manpages and not man/troff
- XMonad.Layout.ImageButtonDecoration & XMonad.Util.Image
- XMonad.Layout.Groups
- XMonad.Layout.ZoomRow
- XMonad.Layout.Renamed
- XMonad.Layout.Drawer
- XMonad.Layout.FullScreen
- XMonad.Hooks.ScreenCorners
- XMonad.Actions.DynamicWorkspaceOrder
- XMonad.Actions.WorkspaceNames
- XMonad.Actions.DynamicWorkspaceGroups

Binary packages of XMonad and XMonadContrib are available for all major Linux 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⟩

7.8.4 Biohaskell

Report by:	Ketil Malde
Participants:	Christian Höner zu Siederdissen, Nick Ignolia

Bioinformatics in Haskell is a steadily growing field, and the *Bio* section on Hackage now sports several libraries and applications. In the past year, we have started to accumulate information related to biohaskell on the biohaskell web site and encourage anyone interested to contribute, and also to sign up to the mailing list.

There are now several libraries for bioinformatics in Haskell, each covering different aspects of the field. These include the *biolib library* that supports various sequence and alignment-oriented file formats and operations, seqloc providing functionality for manipulating sequence locations and annotation, Biobase for working with RNA secondary structure, and samtools wrapping the samtools C library for accessing and manipulating BAM alignment files.

Further reading

- http://biohaskell.org
- http://blog.malde.org/
- http://www.tbi.univie.ac.at/~choener/Haskell/

7.8.5 Bullet

Report by:	Csaba Hruska
Status:	experimental, active development

Bullet is a professional open source multi-threaded 3D Collision Detection and Rigid Body Dynamics Library written in C++. It is free for commercial use under the zlib license. The Haskell bindings ship their own (auto-generated) C compatibility layer, so the library can be used without modifications. The Haskell binding provides a low level API to access Bullet C++ class methods. Some bullet classes (Vector, Quaternion, Matrix, Transform) have their own Haskell representation, others are binded as class pointers. The Haskell API provides access to some advanced features, like constraints, vehicle and more.

At the current state of the project most common services are accessible from Haskell, i.e., you can load collision shapes and step the simulation, define constraints, create raycast vehicle, etc. More advanced Bullet features (soft body simulation, Multithread and GPU constaint solver, etc.) will be added later.

Further reading

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

7.8.6 Sloth2D

Report by:	Patai Gergely
Status:	experimental, active

Sloth2D is a purely functional 2D physics library with composable high-level abstractions. The primary intent behind this initiative is not to compete with existing engines, but rather to experiment with novel, composable abstractions for physics. This might eventually lead to better high-level interfaces for existing engines, e.g., the Chipmunk and Bullet bindings

(→ 7.8.5). However, in the long run it might grow into something that is usable in practice by itself.

The cabalised source is available on GitHub.
Current features:

- 100% pure implementation
- deterministic simulation (replayable regardless of sampling rate)
- convex colliders

Planned features:

- other collider shapes: concave, round, half-plane
- collision layers
- spatial hashing for more efficient collision detection
- object deactivation
- support for raycasting
- serialisation of physics state
- combinators on dynamic worlds
- constraints
- friction
- stacking
- a scene graph-based interface to define the world in a compact manner

Further reading

https://github.com/cobbpg/sloth2d

7.8.7 hledger

Report by:	Simon Michael
Status:	ongoing development; suitable for daily use

hledger is a Haskell port and friendly fork of John Wiegley's ledger. It is a robust command-line accounting tool with a simple human-editable data format. Given a plain text file describing transactions, of money or any other commodity, hledger will print the chart of accounts, account balances, or transactions you are interested in. It can also help you record transactions, or convert CSV data from your bank. There are also curses and web interfaces. The project aims to provide a reliable, practical day-to-day financial reporting tool, and also a useful library for building financial apps in Haskell.

Since hledger's last HCAR entry in 2009, hledger became cabalised, had 10 non-bugfix releases on Hackage, split into multiple packages, acquired a public mailing list, bug tracker, fairly comprehensive manual, cross-platform binaries, and has grown to 5k lines of code and 15 committers. 0.14 has just been released, with 5 code committers.

The project is available under the GNU GPLv3 or later, at http://hledger.org.

Current plans are to continue development at a steady pace, to attract more developers, and to become more useful to a wider range of users, e.g., by building in more awareness of standard accounting procedures and by improving the web and other interfaces.

Further reading

http://hledger.org

7.8.8 arbtt

Report by:	Joachim Breitner
Status:	working

See: http://www.haskell.org/communities/11-2010/html/report.html#sect9.8.5.

7.8.9 uacpid (Userspace ACPI Daemon)

Report by:	Dino Morelli
Status:	experimental, actively developed

See: http://www.haskell.org/communities/05-2010/html/report.html#sect6.13.5.

7.8.10 epub-tools (Command-line epub Utilities)

Report by:	Dino Morelli
Status:	stable, actively developed

A suite of command-line utilities for creating and manipulating epub book files. Included are: epubmeta, epubname, epubzip.

epub-tools is available from Hackage, the Darcs repository below, and also in binary form for Arch Linux through the AUR.

Further reading

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

8 Commercial Users

8.1 Well-Typed LLP

Report by:	Ian Lynagh
Participants:	Duncan Coutts, Andres Löh, Spencer Janssen, Eric Kow, Bernie Pope

Well-Typed is a Haskell services company. We provide commercial support for Haskell as a development platform, including consulting services, training, and bespoke software development. For more information, please take a look at our website or drop us an e-mail at ⟨info@well-typed.com⟩.

Following our second recruiting effort we are delighted that Spencer Janssen is now working full-time for Well-Typed, and Eric Kow and Bernie Pope are also working part-time for us.

On the GHC support contract we have been pleased to be part of the three GHC 7.0 releases since the last HCAR, and are now working towards the upcoming 7.2.1 release. Meanwhile the Parallel GHC Project (→4.1.5) is starting to gather momentum, and the Industrial Haskell Group (→8.3) is taking a look at making some improvements to Cabal. Of course, we also have an amount of proprietary work that is not so visible, and an encouraging number of interesting possibilities on the horizon.

Further reading

◦ http://www.well-typed.com/
◦ Blog: http://blog.well-typed.com/

8.2 Bluespec Tools for Design of Complex Chips and Hardware Accelerators

Report by:	Rishiyur Nikhil
Status:	commercial product

See: http://www.haskell.org/communities/05-2010/html/report.html#sect7.2.

8.3 Industrial Haskell Group

Report by:	Andres Löh
Participants:	Duncan Coutts, Ian Lynagh, Spencer Janssen

The Industrial Haskell Group (IHG) is an organization to support the needs of commercial users of Haskell.

In the past six months, the collaborative development scheme funded work on a library "safeint" for integers that throw an exception once an overflow occurs. Currently, we are making some changes to the Cabal dependency solver, in order to increase the success rate and improve error messages. Details of the tasks undertaken are appearing on the Well-Typed (→8.1) blog and on the IHG status page.

The collaborative development scheme is now running continuously, so if you are interested in joining as a member, please get in touch. Details of the different membership options (full, associate, or academic) can be found on the website.

If you are interested in joining the IHG, or if you just have any comments, please drop us an e-mail at ⟨info@industry.haskell.org⟩.

Further reading

◦ http://industry.haskell.org/
◦ http://industry.haskell.org/status
◦ http://hackage.haskell.org/package/safeint

8.4 Tsuru Capital

Report by:	Bryan Buecking

Tsuru Capital is engaged in high-frequency market-making on options markets. Tsuru is a private company, and trades with its own capital. Tsuru Capital currently runs arbitrage based liquidity provision strategies on the Kospi 200 index and plans to expand to Nikkei 225 index, and other electronic markets, over the next year.

The trading software has been developed entirely in Haskell, and is one of the few systems in the world written completely in a functional language.

Since 2010 we have opened our doors to students, post graduates, and anyone looking for real world experience. And continue to do so by offering paid 3 month internship positions every quarter.

Over the past year we have spent a good deal of time building GUIs for our trading system, and tools for logging and playback. As a result we have contributed bits and pieces of our work to Hackage, and will continue to do so as we flesh out our framework.

Further reading

http://www.tsurucapital.com/

8.5 Oblomov Systems

Report by:	Martijn Schrage

See: http://www.haskell.org/communities/05-2010/html/report.html#sect7.7.

9 Research and User Groups

9.1 Haskell at Eötvös Loránd University (ELTE), Budapest

Report by:	Péter Diviánszky
Status:	ongoing

Haskell and Agda courses:

- Since 2006 Haskell is an option to implement the exercises for the course "Programming Language Concepts of Functional Programming" (the other option is Clean).
- Since 2008 first semester BSc students may learn Haskell (30–40 students per year).
- Since 2009 Haskell is in the curriculum of BSc students specialized in software development. This year more then two hundred BSc students are taught Haskell in their second semester. We have an online evaluation and testing system with a collection of several hundred systematized exercises. I experiment with online graphical exercises with SVG graphics too; I plan to write a blog entry about it soon.
- Since 2009 advanced Haskell is tought for 5–30 master students per semester. Ongoing work is to extend the online interpreter and testing environment with safe emulation of IO values.
- This spring there is an English Haskell course also. I began to translate our course materials to English.
- This spring an Agda course is held for approximately 10 master students for the first time.
- Other Haskell related courses are Lambda Calculus, Type Theory and Implementation of Functional Languages.

Ongoing projects using Haskell:

- Feldspar, a high-level domain-specific language for digital signal processing developed for Ericsson in cooperation with Chalmers University of Technology. Our task is to implement an efficient multi-platform ISO C99 code generator for the language.
- Software Technologies for Distributed and Manycore Systems started in 2010. The Project is supported by the European Union and co-financed by the European Social Fund.

Further reading

- Haskell and Agda course materials (in Hungarian): http://pnyf.inf.elte.hu/fp/
- Parts of course materials in English: http://pnyf.inf.elte.hu/fp/Index_en.xml#course-material
- Feldspar project homepage: http://feldspar.inf.elte.hu

9.2 Functional Programming at UFMG and UFOP

Report by:	Carlos Camarão
Participants:	Marco Gontijo, Rafael Alcântara de Paula, Lucília Figueiredo, Rodrigo Ribeiro, Cristiano Vasconcellos, Elton Ribeiro
Status:	active development

The Functional Programming groups at Universidade Federal de Minas Gerais and Universidade Federal de Ouro Preto work on several projects:

Proposal for a Solution to Haskell's Multi-parameter Type Class Dilemma The introduction of multi-parameter type classes in Haskell has been hindered because of problems associated to ambiguity and inference of uninformative types. We propose a simple solution to this problem, which does not require the use of functional dependencies between type class parameters nor any other extra mechanism, such as type families. A relatively small change to the language is proposed, in order to deal with the problem of inference of uninformative types, and a small change to the type inference algorithm and to what has been considered ambiguity in Haskell is suggested, in order to tackle problems associated to ambiguity. The proposal is described in our SBLP'2009 paper (see below). A related message has been sent to Haskell-cafe and Haskell-prime. We are currently working on an implementation of Haskell's front-end that can type all existing Haskell libraries, including those that require GHC's extensions related to overloading.

Decidable type inference for Haskell overloading We have designed what we consider to be a nice termination criterion for Haskell's type inference algorithm, with respect to overloading (that deals with all the "complicated cases" given in the PPDP'04 and ACM TOPLAS 2005 references below). When types have constraints, decidability of type inference is based mainly on decidability of constraint-set satisfiability. We have designed a constraint-set satisfiability algorithm that uses a simple and in practice unrestrictive criterion in order to guarantee termination. A paper is currently being written, and an implementation is available at https://github.com/rodrigogribeiro/core.

First Class Overloading and Intersection Types The Hindley-Milner type system imposes the restriction that function parameters must have monomorphic types. Lifting this restriction and providing system F "first class" polymorphism is clearly desirable, but comes with the difficulty that complete type inference for higher-ranked type systems is undecidable. More practical higher-ranked type systems have been recently proposed, which rely on system F, and require appropriate type annotations for the definition of functions with polymorphic type parameters. However, these type annotations may, in several cases, inevitably disallow some possible uses of defined higher-rank functions. To avoid this problem and to promote code reuse, we propose the use of intersection types for specifying the types of function parameters used polymorphically inside a function body, allowing a flexible use of such functions, on applications to both polymorphic or overloaded arguments. A paper has been submitted and the work is currently being implemented in our compiler front-end, available at https://github.com/rodrigogribeiro/core.

Controlling the scope of instances in Haskell Marco Gontijo is working on a language extension for Haskell, as part of his MSc research, oriented by Carlos Camarão, to allow control over the import and export of type class instances between modules. The goals of this extension are: i) allow alternative instances of a class for the same type to be defined and used in different module scopes of a program; ii) eliminate problems associated with orphan instances; iii) avoid pollution of the global scope. An article is currently being written, and Rafael Alcântara de Paula is implementing the proposal in a Haskell compiler prototype. The prototype, written by Rodrigo Ribeiro, is available at https://github.com/rodrigogribeiro/core.

Further reading

- *A Solution to Haskell's Multi-paramemeter Type Class Dilemma*, Carlos Camarão, Rodrigo Ribeiro, Lucília Figueiredo, Cristiano Vasconcellos, SBLP'2009 (13th Brazilian Symposium on Programming Languages). http://www.dcc.ufmg.br/~camarao/CT/solution-to-mptc-dilemma.pdf
- *Constraint-set satisfiability for Overloading*, Carlos Camarão, Lucília Figueiredo, Cristiano Vasconcellos, ACM Press Conf. Proceedings of PPDP'04 , 67–77, 2004. http://www.dcc.ufmg.br/~camarao/CT/cs-sat/cssat.pdf
- *A theory of overloading*, Peter J. Stuckey, Martin Sulzmann, ACM TOPLAS 2005, 27(6), 1216–1269. http://portal.acm.org/citation.cfm?id=1108974

9.3 Artificial Intelligence and Software Technology at Goethe-University Frankfurt

Report by:	David Sabel
Participants:	Altug Anis, Conrad Rau, Manfred Schmidt-Schauß

Programming language semantics. One of our research topics focuses on programming language semantics, especially on contextual equivalence which is usually based on the operational semantics of the language.

Deterministic call-by-need lambda calculi with letrec provide a semantics for the core language of Haskell. For such an extended lambda calculus we proved correctness of strictness analysis using abstract reduction, and we proved equivalence of the call-by-name and call-by-need semantics.

We also explored several nondeterministic extensions of call-by-need lambda calculi and their applications. A recent result is that for calculi with `letrec` and nondeterminism usual definitions of similarity are unsound w.r.t. contextual equivalence.

Most recently we analyzed the semantics of a higher-order functional language with concurrent threads, monadic IO and synchronizing variables as in Concurrent Haskell. To assure declarativeness of concurrent programming we extended the language by implicit, monadic, and concurrent futures. Using contextual equivalence based on may- and should-convergence, we established a context lemma and have shown that various transformations preserve program equivalence, e.g. the monad laws hold in our calculus. We also proved that call-by-need and call-by-name evaluation are equivalent, since they induce the same program equivalence.

All these investigations on contextual equivalence have in common that they require to analyze the overlappings between reductions of the operational semantics and transformation steps. In a recent research project we try to automatize correctness proofs of program transformations. A main step for this goal is the computation of overlappings between reductions of the operational semantics and transformations steps. This computation requires the combination of several unification algorithms. We implemented a prototype of this combined algorithm in Haskell.

Grammar based compression. Another research topic of our group focuses on algorithms on grammar compressed strings and trees. One goal is to reconstruct known algorithms on strings and terms (unification, matching, rewriting etc.) for their use on grammars without prior decompression. We recently developed an algorithm for computing the congruence closure on grammar compressed terms. We implemented several algorithms in Haskell and currently prepare a Cabal package containing these algorithms.

Further reading

http://www.ki.informatik.uni-frankfurt.de/research/HCAR.html

9.4 Functional Programming at the University of Kent

Report by: Olaf Chitil

The Functional Programming group at Kent is a subgroup of the Programming Languages and Systems Group of the School of Computing. 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.

Our members pursue a variety of Haskell-related projects, some of which are reported in other sections of this report. Simon Thompson completed a third edition of his Haskell text book, to appear in June 2011. The Haskell Refactorer Hare (→ 5.1.5) has been cabal-ised, and now features clone detection and elimination facilities. Thomas Schilling is developing ideas for improving type error messages for GHC and on trace-based dynamic optimisations for Haskell programs. Olaf Chitil is working on better lazy assertions for Haskell. Neil Brown presented his *Combinators for Message-Passing in Haskell* at PADL 2011 and Olaf Chitil presented a *Semantics for Lazy Assertions* at PEPM 2011.

Further reading

- PLAS group: http://www.cs.kent.ac.uk/research/groups/plas/
- Refactoring Functional Programs: http://www.cs.kent.ac.uk/research/groups/plas/hare.html
- Tracing and debugging with Hat: http://www.haskell.org/hat
- Heat: http://www.cs.kent.ac.uk/projects/heat/
- Scion: http://code.google.com/p/scion-lib/

9.5 Formal Methods at DFKI and University Bremen

Report by:	Christian Maeder
Participants:	Mihai Codescu, Dominik Dietrich, Christoph Lüth, Till Mossakowski, Lutz Schröder, Ewaryst Schulz
Status:	active development

The activities of our group center on formal methods, covering a variety of formal languages and also translations and heterogeneous combinations of these.

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 the Common Algebraic Specification Language (CASL) itself (which provides many-sorted first-order logic with partiality, subsorting and induction), HasCASL, CoCASL, CspCASL, and ModalCASL. Other languages supported include Haskell (via Programatica), QBF, Maude, VSE, TPTP (THF is on the way), OWL, Common Logic, FPL (logic of functional programs) and LF type theory. The Hets implementation is also based on some old Haskell sources such as bindings to uDrawGraph (formerly Davinci) and Tcl/TK that we maintain. Apart from a gtk2hs user interface hets also provides many functionalities as a web server based on wai (wai-extra-0.2.x).

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.

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/
- The Coalgebraic Logic Satisfiability Solver CoLoSS:
  http://www.informatik.uni-bremen.de/~lschrode/

projects/GenMod
http://www.doc.ic.ac.uk/~dirk/COLOSS/

9.6 Haskell at Universiteit Gent, Belgium

Report by: Tom Schrijvers

Haskell is one of the main research topics of the new Programming Languages Group at the Department of Applied Mathematics and Computer Science at the University of Ghent, Belgium.

Haskell-related projects of the group members and collaborators are:

∘ *Search Combinators:* Search heuristics often make all the difference between effectively solving a combinatorial problem and utter failure. Hence, the ability to swiftly design search heuristics that are tailored towards a problem domain is essential to performance improvement. In other words, this calls for a high-level domain-specific language (DSL).

  The tough technical challenge we face when designing a DSL for search heuristics, is to bridge the gap between a conceptually simple specification language (high-level, purely functional and naturally compositional) and an effecient implementation (typically low-level, imperative and highly non-modular). We overcome this challenge with a systematic approach in Haskell that disentangles different primitive concepts into separate monadic modular mixin components, each of which corresponds to a feature in the high-level DSL. The great advantage of mixin components to provide a semantics for our DSL is its modular extensibility.

  This is joint work with Guido Tack, Pieter Wuille, Horst Samulowitz and Peter Stuckey, following up on *Monadic Constraint Programming*, a monadic DSL for Constraint Programming in Haskell.

∘ *Monads, Zippers and Views: Virtualizing the Monad Stack*: We make monadic components more reusable and robust to changes by employing two new techniques for *virtualizing* the monad stack: the *monad zipper* and *monad views*. The monad zipper is a higher-order monad transformer that creates virtual monad stacks by ignoring particular layers in a concrete stack. Monad views provide a general framework for monad stack virtualization: they take the monad zipper one step further and integrate it with a wide range of other virtualizations. For instance, particular views allow restricted access to monads in the stack. Furthermore, monad views provide components with a *call-by-reference*-like mechanism for accessing particular layers of the monad stack. With our two new mechanisms, the monadic effects required by components no longer need to be literally reflected in the concrete monad stack. This makes these components more reusable and robust to changes.

  This is joint work with Bruno Oliveira, part of which is available together with Mauro Jaskelioff's monad transformer library in the Monatron package on Hackage.

∘ *Type Checking:* The latest result is OUTSIDEIN(X), a framework for modular type inference with local assumptions. Earlier results are on type inference for GADTs, type invariants, and type checking for type families. 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, Manuel Chakravarty, Dimitrios Vytiniotis, Stefan Monnier, Louis-Julien Guillemette, and Dominic Orchard.

∘ *EffectiveAdvice:* EffectiveAdvice is a disciplined model of (AOP-style) advice, inspired by Aldrich's Open Modules, that has full support for effects in both base components and advice. EffectiveAdvice is implemented as a Haskell library. Advice is modeled by mixin inheritance and effects are modeled by monads. Interference patterns previously identified in the literature are expressed as combinators. Equivalence of advice, as well as base components, can be checked by equational reasoning. Parametricity, together with the combinators, is used to prove two harmless advice theorems. The result is an effective model of advice that supports effects in both advice and base components, and allows these effects to be separated with strong non-interference guarantees, or merged as needed. This is joint work with Bruno Oliveira and William Cook.

We are also involved in the organization of the Ghent Functional Programming Group (→ 9.13).

Further reading

∘ http://users.ugent.be/~tschrijv/haskell.html
∘ http://users.ugent.be/~tschrijv/SearchCombinators/
∘ http://hackage.haskell.org/package/Monatron
∘ http://hackage.haskell.org/package/monadiccp

9.7 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 as time goes. Actually, the Ro/Haskell Group is officially a project of the Faculty of Sciences, "V. Alecsandri" Univ. of Bacău, România (http://stiinte.ub.ro) based by volunteers.

Website:

On the 9th of May 2011, the main Ro/Haskell's web page counter recorded the total of 36,500 times accessed. The movement of the website from one server to another had broken a set of links. Even one from Wikipedia was broken, leading to the rejection of a page concerning Haskell as undocumented. The page was rebuilt. Also, after some arguments, the Romanian Wikipedia Site hosted a page concerning Haskell, even if some Universities and results was deleted. On the other side we are in the process of tracking papers of Romanian authors and link them to the Ro/Haskell website. Some of them was on the servers of their coauthors, outside of Romania, being difficult to find.

Books:

The book "The Practice Of Monadic Interpretation" by Dan Popa had been published in November 2008. The book had developed into a full PhD. thesis which was successfully defended in public in September 2010. Also notice that the page of the book was in the top ten, based on access rate: 12,225 times. No English version is available so far. Any editor interested? Actually the Official Publishing House of the Ro/Haskell Group is MatrixRom (www.matrixrom.ro). Speaking of books, the "Gentle introduction to Haskell" is prepairing to be released in a Romanian translation. The introductory chapter (http://www.haskell.org/wikiupload/3/38/Gentle_1-19-v06-3Aprilie.pdf.zip) can be downloaded from http://www.haskell.org/haskellwiki/Gentle where two other versions are available, too: French and of course English. "An Introduction to Haskell by Examples" is now out of print but if you need, a special pack can be provided based on the agreement of the author ⟨popavdan@yahoo.com⟩.

Products:

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. Rodin was asked to become a FOSS (Free & Open Source Software) and will be. To have a sort of C using native keywords was a success in teaching basics of Computer Science: algorithms and structured programming. As a consequence of having such a DSL and its website, a course in Fundamentals of Computers Science was reduced to 4 hours instead of an entire semester. As a consequence, The Web Page of the Rodin DSL had 10,009 hits this mounth (May 2011).

Linguists:

A group of researchers from the field of linguistics located at the State Univ. from Bacău (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. We had begun by publishing an article about Parser Combinators in a volume (Studii si Cercetari Stiintifice — Seria Filologie 23/2010, Ed. Alma Mater, Bacău — the volume is ready for print, after a year of work.) and provide information that Haskell can be used for e-learning and e-manuals of foreign languages.

At Bacău "V. Alecsandri" University

We have teaching Haskell at two 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. 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.

Notions:

We are promoting new notions: pseudoconstructors over monadic values (which act both as semantic representations and syntactic structure), modular trees (expanding trees beyound the fixity of the data declarations) and ADFA — adaptive/adaptable determinist finite automata. A dictionary of new notions and concepts is not made, making difficult to launch new ideas and also to track work of the authors.

Unsolved problems:

PhD. advisors (specialized in monads, language engineering, and Haskell) are almost impossible to find. This fact seems to block somehow the hiring of good specialists in Haskell. Also it is difficult to track the Haskell related activity from various universities, like those from: Sibiu, Baia Mare, Timisoara. Please report them using the below address.

Contact

⟨popavdan@yahoo.com⟩

Further reading

- Ro/Haskell: http://www.haskell.org/haskellwiki/Ro/Haskell
- Rodin: http://www.haskell.org/haskellwiki/Rodin

- Gentle introduction to Haskell (Ro): http://www.haskell.org/haskellwiki/Gentle
- ADFA: http://www.haskell.org/haskellwiki/ADFA
- Report from: http://stiinte.ub.ro (the Faculty I belong to)

9.8 fp-syd: Functional Programming in Sydney, Australia

Report by:	Erik de Castro Lopo
Participants:	Ben Lippmeier, Shane Stephens, and others

We are a seminar and social group for people in Sydney, Australia, interested in Functional Programming and related fields. Members of the group include users of Haskell, Ocaml, LISP, Scala, F#, Scheme and others. We have 10 meetings per year (Feb–Nov) and meet on the third Thursday of each month. We regularly get 20–30 attendees, with a 70/30 industry/research split. Talks this year have included material on Category Theory, theorem proving in Coq, Template Haskell and a couple of different Haskell libraries. We usually have about 90 mins of talks, starting at 6:30pm, then go for drinks afterwards. All welcome.

Further reading

- http://groups.google.com/group/fp-syd
- http://fp-syd.ouroborus.net/

9.9 Functional Programming at Chalmers

Report by:	Jean-Philippe Bernardy

Functional Programming is an important component of the Department of Computer Science and Engineering at Chalmers. In particular, Haskell has a very important place, as it is used as the vehicle for teaching and numerous projects. Besides functional programming, language technology, and in particular domain specific languages is a common aspect in our projects.

Property-based testing QuickCheck is the basis for a European Union project on Property Based Testing (www.protest-project.eu). We are applying the QuickCheck approach to Erlang software, together with Ericsson, Quviq, and others. Much recent work has focused on PULSE, the ProTest User-Level Scheduler for Erlang, which has been used to find race conditions in industrial software — see our ICFP 2009 paper for details. A new tool, QuickSpec, generates algebraic specifications for an API automatically, in the form of equations verified by random testing. We have published about it at TAP 2010; an earlier paper can be found here: http://www.cse.chalmers.se/~nicsma/quickspec.pdf. Lastly, we have devised a technique to speed up testing of polymorphic properties: http://publications.lib.chalmers.se/cpl/record/index.xsql?pubid=99387.

Natural language technology Grammatical Framework (→ 7.7.3) is a declarative language for describing natural language grammars. It is useful in various applications ranging from natural language generation, parsing and translation to software localization. The framework provides a library of large coverage grammars for currently fifteen languages from which the developers could derive smaller grammars specific for the semantics of a particular application.

Parser generator and template-haskell BNFC-meta is a parser generator. Like the BNF Converter, it generates a compiler front end in Haskell. Two things separate BNFC-meta from BNFC and other parser generators:
- BNFC-meta is not a program but a library (the parser description is embedded in a quasi-quote).
- BNFC-meta automatically provides quasi-quotes for the specified language. This includes a powerful and flexible facility for antiquotation.

More info: http://hackage.haskell.org/package/BNFC-meta.

Generic Programming Starting with Polytypic Programming in 1995 there is a long history of generic programming research at Chalmers. Recent developments include fundamental work on parametricity & dependent types (ICFP 2010), a survey paper "Generic programming with C++ concepts and Haskell type classes" (JFP 2010) and two new PhD students. Patrik Jansson leads a work-package on DSLs within the EU project "Global Systems Dynamics and Policy" (http://www.gsdp.eu/, started Oct. 2010). If you want to apply DSLs, Haskell, and Agda to help modelling global sustainability challenges, please get in touch!

Language-based security SecLib is a light-weight library to provide security policies for Haskell programs. The library provides means to preserve confidentiality of data (i.e., secret information is not leaked) as well as the ability to express intended releases of information known as declassification. Besides confidentiality policies, the library also supports another important aspect of security: integrity of data. SecLib provides an attractive, intuitive, and simple setting to explore the security policies needed by real programs.

Type theory Type theory is strongly connected to functional programming research. Many dependently-typed programming languages and type-based proof assistants have been developed at Chalmers. The Agda system (→ 3.1) is the latest in this line, and is of particular interest to Haskell programmers. We encourage you to experiment with programs and proofs in Agda as a “dependently typed Haskell”.

DSP programming Feldspar is a domain-specific language for digital signal processing (DSP), developed in co-operation by Ericsson, Chalmers FP group and Eötvös Loránd (ELTE) University in Budapest. The motivating application is telecom processing, but the language is intended to be more general. As a first stage, we have focused on the data-intensive numeric algorithms which are at the core of any DSP application. More recently, we have started to work on extending the language to deal with more system-level aspects. The data processing language is purely functional and highly inspired by Haskell. Currently the language is implemented as an embedded language in Haskell.

The implementation is available from Hackage: http://hackage.haskell.org/package/feldspar-language. There is also a code generator, developed at ELTE University: http://hackage.haskell.org/package/feldspar-compiler.

See also the official project page: http://feldspar.inf.elte.hu.

Hardware design/verification The functional programming group has developed three different hardware description languages — Lava, Wired, and Chalk (chronological order) — implemented in Haskell. Each language targets a different abstraction level. The basic idea behind all three is to model circuits as functions from inputs to outputs. This allows structural hardware description in standard functional programming style.

Chalk is a new language for architecture design. Once you have defined a Chalk circuit, you can simulate it, or explore it further using non-standard interpretations. This is particularly useful if you want to perform high-level power and performance analysis early on in the design process.

More info: http://www.cse.chalmers.se/~wouter/Publications/DCC2010.pdf.

In **Lava**, circuits are described at the gate level (with some RTL support). The version developed at Chalmers has a particular aim to support formal verification in a convenient way.

Wired is an extension to Lava, targeting (not exclusively) semi-custom VLSI design. A particular aim of Wired is to give the designer more control over on-chip wires' effects on performance. Some features of Wired are:

- Initial description can be purely functional (a la Lava).
- Incremental specification of physical aspects.
- Accurate, wire-aware timing/power analysis within the system.
- Support for an academic 45nm cell library.

Wired is not actively developed at the moment, but the system has recently been used to explore the layout of multipliers (Kasyab P. Subramaniyan, Emil Axelsson, Mary Sheeran and Per Larsson-Edefors. Layout Exploration of Geometrically Accurate Arithmetic Circuits. *Proceedings of IEEE International Conference of Electronics, Circuits and Systems.* 2009).

Home page: http://www.cse.chalmers.se/~emax/wired/.

Automated reasoning Equinox is an automated theorem prover for pure first-order logic with equality. Equinox actually implements a hierarchy of logics, realized as a stack of theorem provers that use abstraction refinement to talk with each other. In the bottom sits an efficient SAT solver. Paradox is a finite-domain model finder for pure first-order logic with equality. Paradox is a MACE-style model finder, which means that it translates a first-order problem into a sequence of SAT problems, which are solved by a SAT solver. Infinox is an automated tool for analyzing first-order logic problems, aimed at showing finite unsatisfiability, i.e., the absence of models with finite domains. All three tools are developed in Haskell.

Teaching Haskell is present in the curriculum as early as the first year of the Bachelors program. We have three courses solely dedicated to functional programming (of which two are Masters-level courses), but we also provide courses which use Haskell for teaching other aspects of computer science, such as programming languages, compiler construction, hardware description and verification, data structures and programming paradigms.

9.10 Functional Programming at KU

Report by:	Andy Gill
Status:	ongoing

Functional Programming remains active at KU and the Computer Systems Design Laboratory in ITTC. The System Level Design Group (lead by Perry Alexander) and the Functional Programming Group (lead by

Andy Gill) together form the core functional programming initiative at KU. Apart from Kansas Lava (→ 7.5.3) and ChalkBoard (→ 6.7.6), there are many other FP and Haskell related things going on.

- We are developing a Haskell version of HOL. Traditionally, members of the higher-order logic theorem (HOL) proving family have been implemented in the Standard ML programming language or one of its derivatives. HaskHOL aims to break with tradition by implementing a lightweight HOL theorem prover library as a Haskell hosted domain specific language. Based on the HOL Light logical system, HaskHOL aims to provide the ability for Haskell users to reason about their code directly without having to transform it or otherwise export it to an external tool. For details talk to Evan Austin.

- We are actively working on enabling *Type-Directed Specification Refinement in Rosetta*. Rosetta is a specification language that focuses on the interaction between different domains, such as state-based and signal-based domains. With dependent types, first-class types, and reflection, there are many areas where a traditional all-or-nothing typing analysis would be impractical — especially when considering that specifications are likely written at first in a high-level, incomplete fashion. This project uses InterpreterLib (http://haskell.org/communities/11-2008/html/report.html#sect5.5.6) and various Rosetta analysis tools to define a typing analysis that attempts to extract typing information, constraints, and errors to present to the user, in order to guide the specification refinement process. It is in the early stages of development, but may eventually link up with HaskHOL to discharge some TCC's. For details talk to Mark Snyder.

- We are developing a library in Haskell for processing Rosetta specifications. A current focus is the modularity and re-use of distinct processing elements, such as type-checking, partial evaluation, and reasoning assistants. Mutually defined elements that are more convenient to consider as distinct interact via a reactive monadic computation, so the two elements' code can be managed as separate packages. Also, our principal specification representation use functors and type-level fixed points to achieve extensibility and generic programming. The goal of the library is to provide to a tight and graduated interface to the basic processing elements, so that the users may incorporate the most appropriate basic elements when implementing their own, more domain-specific Rosetta processors. For details talk to Nick Frisby.

- We are working with other functional programming groups (University of Iowa, St. Andrews, Heriot-Watt, Halmstad University, and of course Chalmers) to share our common experiences with using FPGA boards, and generating VHDL. So far, we have chosen and purchased common Xilinx boards, and have a design for a so called "λ-bridge" between our UNIX invocation infrastructures and our FPGA boards. The idea is we can share experiences, for the sake of being able to spend more time working on FP issues, and bringing FP ideas to hardware related problems.

Further reading

- The Functional Programming Group: http://www.ittc.ku.edu/csdl/fpg
- CSDL website: https://wiki.ittc.ku.edu/csdl/Main_Page

9.11 Dutch Haskell User Group

Report by:	Tom Lokhorst

The Dutch Haskell User Group is a diverse group of people interested in Haskell and functional programming.

Our group was founded in April of 2009, at the 5th Haskell Hackathon in Utrecht. Since then, we have had monthly meetings and an afternoon symposium. Our meetings alternate between pure socializing and evenings that include talks by members.

We cater to the hobbyist, the academic, and the professional crowds. Anyone is welcome to join, from beginners to advanced users. Do join us!

Further reading

http://dutchhug.nl/

9.12 San Simón Haskell Community

Report by:	Antonio M. Quispe

The San Simón Haskell Community from San Simón University Cochabamba-Bolivia, is an informal Spanish group that aspire to learn, share information, knowledge and experience related to the functional paradigm.

Our main activity is the development of projects, we have some projects in our Web Page (http://comunidadhaskell.org) that serves us as a medium of communication and work environment.

Our last activity was the Local Haskell Hackathon that was held on April 8, 9 and 10 in our University. There were 15 participants of different levels in functional programming. We have been working on projects idbjava (decompiler bytecode java), lexer and parser for Ruby, emulator for CNC machine, and some Haskell games. We have had a wonderful time of 2 days of programming, and I want to thank Vladimir Costas and Pablo Azero for their assistence in the realization of this event.

The next thing we are waiting on is the 2nd Open House Haskell community where we will show some of the projects we are working on.

I want to encourage all Spanish Haskell programmers to meet us on Facebook.

Further reading

http://comunidadhaskell.org

9.13 Ghent Functional Programming Group

Report by:	Jeroen Janssen
Participants:	Bart Coppens, Jasper Van der Jeugt, Tom Schrijvers, Andy Georges, Kenneth Hoste
Status:	active

GHENTFPG

The Ghent Functional Programming Group is a new user group aiming to bring together programmers, academics, and others interested in functional programming located in the area of Ghent, Belgium. Our goal is to have regular meetings with talks on functional programming, organize functional programming related events such as hackathons, and to promote functional programming in Ghent by giving after-hours tutorials.

The first five GhentFPG meetings and BelHac were reported on in the previous HCARs. Since then we have held two other GhentFPG meetings. GhentFPG #6, held in February 2011, was a problem-solving night where we tackled some hard problems in Haskell. GhentFPG #7 was a regular meeting with the following three talks:

1. Tom Van Custem — Experiments with MapReduce in Erlang. MapReduce is a programming model for large data processing popularized by, and in daily use at Google. The MapReduce model builds strongly on key tenets of functional programming such as higher-order functions and side-effect free execution. In this talk, we summarize this programming model and describe a didactic implementation in Erlang. Invented at Ericsson's research labs, Erlang is known for its massively concurrent programming model, and itself builds on a functional core language. The talk will not focus on Erlang as such, but we will describe its key features as needed to understand the MapReduce abstraction.

2. Tom Schrijvers — How you could have won the VPW 2011 contest with Haskell. We all know that Functional Programming is great for writing concise solutions for programming problems. With some skill this can even be done quickly! Yet, there was little evidence of this at the 3rd edition of the Flemish Programming Contest (VPW 2011) that took place on March 23. Not so before the contest: The jury stress-tested all questions by writing various solutions in different languages. Haskell was used to solve most problems and invariably produced short solutions.

   In this talk I present my own Haskell solutions to several of this year's problems and discuss alternative solution strategies with the audience. After the talk you will be all set for winning next year's edition — or at least enjoying it — using Haskell.

3. Pieter Audenaert — Functional Geometry and a Graphical Language. We will discuss a simple language for drawing images. During the presentation we will illustrate the power of data abstraction and algebraic closure, meanwhile using higher order procedures in an essential manner. The language has been designed to easy experimenting with patterns such as those appearing in typical M.C. Escher drawings where the artist repeats the pattern both moving it across the drawing and scaling it when applicable. In the language we use procedures to represent the data objects that will be combined in the final drawing and we make sure that all operations conducted on these procedures are algebraically closed. These features allow generating patterns of any complexity.

   For our implementation, we use the LISP functional programming language — more accurately, the Scheme dialect. The presentation is based on "Structure and Interpretation of Computer Programs", Abelson & Sussman

If you want more information on GhentFPG you can follow us on twitter (@ghentfpg), via Google Groups (http://groups.google.com/group/ghent-fpg), or by visiting us at irc.freenode.net in channel #ghentfpg.

Further reading

http://groups.google.com/group/ghent-fpg