Haskell Communities and Activities Report

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

Sixteenth Edition — May 2009

Janis Voigtländer (ed.)

Peter Achten
Andy Adams-Moran
Lloyd Allison
Tiago Miguel Laureano Alves
Krasimir Angelov
Heinrich Apfelmus
Emil Axelsson
Arthur Baars
Justin Bailey
Alistair Bayley
Jean-Philippe Bernardy
Clifford Beshers
Gwern Branwen
Joachim Breitner
Niklas Broberg
Björn Buckwalter
Denis Bueno
Andrew Butterfield
Roman Cheplyaka
Olaf Chitil
Jan Christiansen
Sterling Clover
Alberto Gómez Corona
Duncan Coutts
Jácome Cunha
Nils Anders Danielsson
Atze Dijkstra
Robert Dockins
Chris Eidhof
Conal Elliott
Henrique Ferreiro García
Marc Fontaine
Nicolas Frisby
Peter Gavin
Patai Gergely
Brett G. Giles
Andy Gill
George Giorgidze
Dimitry Golubovsky
Daniel Gorin
Jurriaan Hage
Bastiaan Heeren
Claude Heiland-Allen
Aycan Irican
Judah Jacobson
Jan Martin Jansen
Wolfgang Jeltsch
Florian Haftmann
Kevin Hammond
Enzo Haussecker
Christopher Lane Hinson
Guillaume Hoffmann
Martin Hofmann
Creighton Hogg
Csaba Hruska
Liyang HU
Paul Hudak
Graham Hutton
Farid Karimipour
Garrin Kimmell
Oleg Kiselyov
Lennart Kolmodin
Slawomir Kolodynski
Michal Konečný
Eric Kow
Stephen Lavelle
Sean Leather
Huiqing Li
Bas Lijnse
Ben Lippmeier
Andres Löh
Rita Loogen
Ian Lynagh
John MacFarlane
Christian Maeder
José Pedro Magalhães
Ketil Malde
Blažević Mario
Simon Marlow
Michael Marte
Bart Massey
Simon Michael
Arie Middelkoop
Ivan Lazar Miljenovic
Neil Mitchell
Maarten de Mol
Dino Morelli
Matthew Naylor
Rishiyur Nikhil
Thomas van Noort
Johan Nordlander
Jeremy O'Donoghue
Patrick O. Perry
Jens Petersen
Simon Peyton Jones
Dan Popa
Fabian Reck
Claus Reinke
Alexey Rodriguez
Alberto Ruiz
David Sabel
Ingo Sander
Uwe Schmidt
Martijn Schrage
Tom Schrijvers
Paulo Silva
Ben Sinclair
Ganesh Sittampalam
Martijn van Steenbergen
Dominic Steinitz
Don Stewart
Jon Strait
Martin Sulzmann
Doaitse Swierstra
Wouter Swierstra
Hans van Thiel
Henning Thielemann
Wren Ng Thornton
Phil Trinder
Jared Updike
Marcos Viera
Miguel Vilaca
David Waern
Malcolm Wallace
Jinjing Wang
Kim-Ee Yeoh
Brent Yorgey

Preface

This is the 16th edition of the Haskell Communities and Activities Report. There are a number of completely new entries and many small updates. As usual, fresh entries are formatted using a blue background, while updated entries have a header with a blue background. Entries on which no activity has been reported for a year or longer have been dropped. Please do revive them next time if you have news on them.

My aim for October is to significantly increase the percentage of blue throughout the pages of the report. Clearly, there are two ways to achieve this: have more blue or have less white. My preference is the former option, and I count on the community to make it feasible. In any case, I will tend to be less conservative than this time around: I plan to drop, or simply replace with pointers to previous versions, entries that see no update between now and then. A call for new entries and updates to existing ones will be issued on the usual mailing lists.

Now enjoy the current report and see what other Haskellers have been up to in the last half year. Any kind of feedback is of course very welcome ⟨hcar@haskell.org⟩, as are thoughts on the plan outlined above.

Janis Voigtländer, Technische Universität Dresden, Germany

Contents

1 Information Sources

1.1 Book: Programming in Haskell

Report by: Graham Hutton

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

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

Publication details:

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

In-depth review:

- Duncan Coutts, The Monad.Reader (→ 1.2), http://www.haskell.org/sitewiki/images/0/03/TMR-Issue7.pdf

Further reading

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

1.2 The Monad.Reader

Report by: Wouter Swierstra

There are plenty of academic papers about Haskell and plenty of 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 maybe do not warrant an academic publication — but that does not mean these ideas are not worth writing about! Communicating ideas to a wide audience is much more important than concealing them in some esoteric journal. Even if it has all been done before in the Journal of Impossibly Complicated Theoretical Stuff, explaining a neat idea about "warm fuzzy things" to the rest of us can still be plain fun.

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

Since the last HCAR there have been two new issues, including another "Summer of Code" special. With your submissions, I expect we can keep publishing quarterly issues of the Monad.Reader. Check out the Monad.Reader homepage for all the information you need to start writing your article.

Further reading

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

1.3 Haskell Wikibook

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

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

Currently, the wikibook is slowly advancing in breath rather than depth, with a new sketch of a chapter on polymorphism and higher rank types, and a new page on Monoids. Additional authors and contributors that help writing new contents or simply spot mistakes and ask those questions we had never thought of are more than welcome!

Further reading

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

1.4 Monad Tutorial

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

The "Greenhorn's Guide to becoming a Monad Cowboy" is yet another monad tutorial. It covers the basics, with simple examples, and includes monad transformers and monadic functions. The didactic style is a variation on the "for dummies" format. Estimated learning time, for a monad novice, is 2–3 days. It is available at http://www.muitovar.com/monad/moncow.xhtml

Further reading

http://www.muitovar.com/

1.5 Oleg's Mini tutorials and assorted small projects

Report by: Oleg Kiselyov

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

Monadic Regions

Monadic Regions is a technique for managing resources such as memory areas, file handles, database connections, etc. Regions offer an attractive alternative to both manual allocation of resources and garbage collection. Unlike the latter, region-based resource management makes resource disposal and finalization predictable. We can precisely identify the program points where allocated resources are freed and finalization actions are run. Like other automatic resource managements schemes, regions statically assure us that no resource is used after it is disposed of, no resource is freed twice, and all resources are eventually deallocated.

We first describe a very simple implementation of Monadic Regions for the particular case of file IO, statically ensuring that there are no attempts to access an already closed file handle, and all open file handles are eventually closed. Many handles can be open simultaneously; the type system enforces the proper nesting of their regions. The technique has no run-time overhead and induces no run-time errors, requiring only the rank-2–type extension to Haskell 98. The technique underlies safe database interfaces of Takusen.

The simplicity of the implementation comes with a limitation: it is impossible to store file handles in reference cells and return them from an inner region, even when it is safe to do so. We show that practice often requires more flexible region policies.

With Chung-chieh Shan, we developed a novel, improved, and extended implementation of the Fluet and Morrisett's calculus of nested regions in Haskell, with file handles as resources. Our library supports region polymorphism and implicit region subtyping, along with higher-order functions, mutable state, recursion, and run-time exceptions. A program may allocate arbitrarily many resources and dispose of them in any order, not necessarily LIFO. Region annotations are part of an expression's inferred type. Our implementation assures timely deallocation even when resources have markedly different lifetimes and the identity of the longest-living resource is determined only dynamically.

For contrast, we also implement a Haskell library for manual resource management, where deallocation is explicit and safety is assured by a form of linear types. We implement the linear typing in Haskell with the help of phantom types and a parameterized monad to statically track the type-state of resources.

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

Variable (type)state "monad"

The familiar State monad lets us represent computations with a state that can be queried and updated. The state must have the same type during the entire computation however. One sometimes wants to express a computation where not only the value but also the type of the state can be updated – while maintaining static typing. We wish for a parameterized "monad" that indexes each monadic type by an initial (type)state and a final (type) state. The effect of an effectful computation thus becomes apparent in the type of the computation, and so can be statically reasoned about.

In a parameterized monad `m p q a`, `m` is a type constructor of three type arguments. The argument `a` is the type of values produced by the monadic computation. The two other arguments describe the type of the computation state before and after the computation is executed. As in ordinary monad, `bind` is required to be associative and `ret` to be the left and the right unit of `bind`. All ordinary monads are particular case of parameterized monads, when `p` and `q` are the same and constant throughout the computation. Like the ordinary class monad, parameterized monads are implementable in Haskell 98.

The web page describes various applications of the type-state Monad:

- IO-like monad that statically enforces a locking protocol;
- statically tracking ticks so to enforce timing and protocol restrictions when writing device drivers and their generators;
- implementing a form of linear types for resource management.

http://okmij.org/ftp/Computation/monads.html#param-monad

1.6 Haskell Cheat Sheet

Report by: Justin Bailey

The "Haskell Cheat Sheet" covers the syntax, keywords, and other language elements of Haskell 98. It is intended for beginning to intermediate Haskell programmers and can even serve as a memory aid to experts.

The cheat sheet is distributed as a PDF and executable source file in one package. It is available from Hackage and can be installed with cabal.

Further reading

http://cheatsheet.codeslower.com

1.7 The Happstack Tutorial

Report by: Creighton Hogg

The Happstack Tutorial aims to be a definitive, up-to-date, resource for how to use the Happstack libraries. I have recently taken over the project from Thomas Hartman. An instance of the Happstack Tutorial is running as a stand-alone website, but in order to truly dig into writing Happstack applications you can cabal install it from Hackage and experiment with it locally.

Happstack Tutorial is updated along with the Happstack Hackage releases, but the darcs head is generally compatible with the darcs head of Happstack.

I am adding a few small tutorials to the package with every release and am always looking for more feedback from beginning Happstack users.

Further reading

- http://tutorial.happstack.com
- http://patch-tag.com/repo/happstack-tutorial

2 Implementations

2.1 The Glasgow Haskell Compiler

Report by:	Simon Peyton Jones
Participants:	many others

The last six months have been busy ones for GHC.

The GHC 6.10 branch

We finally released GHC 6.10.1 on 4 November 2008, with a raft of new features we discussed in the October 2008 status report.

A little over five months later we released GHC 6.10.2, with more than 200 new patches fixing more than 100 tickets raised against 6.10.1. We hoped that would be it for the 6.10 branch, but we slipped up and 6.10.2 contained a couple of annoying regressions (concerning Control-C and editline). By the time you read this, GHC 6.10.3 (fixing these regressions) should be out, after which we hope to shift all our attention to the 6.12 branch.

The new build system

Our old multi-makefile build system had grown old, crufty, hard to understand. And it did not even work very well. So we embarked on a plan to re-implement the build system. Rather than impose the new system on everyone immediately, Ian and Simon (Marlow) did all the development on a branch, and invited others to give it a whirl. Finally, on 25 April 2009, we went "live" on the HEAD.

The new design is described on the Wiki. It still uses `make`, but it is now based on a non-recursive make strategy. This means that dependency tracking is vastly more accurate than before, so that if something "should" be built it "will" be built.

The new build system is also much less dependent on Cabal than it was before. We now use Cabal only to read package meta-data from the `<pkg>.cabal` file, and emit a bunch of Makefile bindings. Everything else is done in `make`. You can read more about the design rationale on the Wiki.

We also advertised our intent to switch to Git as our version control system (VCS). We always planned to change the build system first, and only then tackle the VCS. Since then, there has been lots of activity on the Darcs front, so it is not clear how high priority making this change is. We would welcome your opinion (⟨cvs-ghc@haskell.org⟩).

The GHC 6.12 branch

The main list of new features in GHC 6.12 remains much the same as it was in our last status report. Happily, there has been progress on all fronts.

Parallel performance

Simon Marlow has been working on improving performance for parallel programs, and there will be significant improvements to be had in 6.12 compared to 6.10. In particular

- There is an implementation of lock-free work-stealing queues, used for load-balancing of sparks and also in the parallel GC. Initial work on this was done by Jost Berthold.
- The parallel GC itself has been tuned to retain locality in parallel programs. Some speedups are dramatic.
- The overhead for running a spark is much lower, as sparks are now run in batches rather than creating a new thread for each one. This makes it possible to take advantage of parallelism at a much finer granularity than before.
- There is optional "eager-blackholing", with the new `-feager-blackholing` flag, which can help eliminate duplicate computation in parallel programs.

Our recent ICFP submission Runtime Support for Multicore Haskell describes all these in more detail, and gives extensive measurements.

Things are not in their final state yet: for example, we still need to work on tuning the default flag settings to get good performance for more programs without any manual tweaking. There are some larger possibilities on the horizon too, such as redesigning the garbage collector to support per-CPU independent GC, which will reduce the synchronization overheads of the current stop-the-world strategy.

Parallel profiling

GHC 6.12 will feature parallel profiling in the form of ThreadScope, under development by Satnam Singh, Donnie Jones and Simon Marlow. Support has been added to GHC for lightweight runtime tracing (work originally done by Donnie Jones), which is used by ThreadScope to generate profiles of the program's real-time execution behavior. This work is still in the very early stages, and there are many interesting directions we could take this in.

Data Parallel Haskell

Data Parallel Haskell remains under very active development by Manuel Chakravarty, Gabriele Keller, Roman Leshchinskiy, and Simon Peyton Jones. The current state of play is documented on the Wiki. We also wrote a substantial paper Harnessing the multicores: nested data parallelism in Haskell for FSTTCS 2008; you may find this paper a useful tutorial on the whole idea of nested data parallelism.

The system currently works well for small programs, such as computing a dot product or the product of a sparse matrix with a dense vector. For such applications, the generated code is as close to hand written C code as GHC's current code generator enables us to be (i.e., within a factor of 2 or 3). We ran three small benchmarks on an 8-core x86 server and on an 8-core UltraSPARC T2 server, from which we derived two comparative figures: a comparison between x86 and T2 on a memory-intensive benchmark (dot product) and a summary of the speedup of three benchmarks on x86 and T2. Overall, we achieved good absolute performance and good scalability on the hardware we tested.

Our next step is to scale the implementation up to properly handle larger programs. In particular, we are currently working on improving the interaction between vectorized code, the aggressively-inlined array library, and GHC's standard optimization phases. The current main obstacle is excessively long compile times, due to a temporary code explosion during optimization. Moreover, Gabriele started to work on integrating specialized support for regular multi-dimensional arrays into the existing framework for nested data parallelism.

Type system improvements

The whole area of GADTs, indexed type families, and associated types remains in a ferment of development. It is clear that type families are jolly useful: many people are using them even though they are only partially supported by GHC 6.10. (You might enjoy a programmers-eye-view tutorial Fun with type functions that Oleg, Ken, and Simon wrote in April 2009.)

But these new features have made the type inference engine pretty complicated, and Simon PJ, Manuel Chakravarty, Tom Schrijvers, Dimitrios Vytiniotis, and Martin Sulzmann have been busy thinking about ways to make type inference simpler and more uniform. Our ICFP'08 paper Type checking with open type functions was a first stab (which we subsequently managed to simplify quite a bit). Our new paper (to be presented at ICFP'09) Complete and decidable type inference for GADTs tackles a different part of the problem. And we are not done yet; for example, our new inference framework is designed to smoothly accommodate Dimitrios' work on FPH: First class polymorphism for Haskell (ICFP'08).

Other developments

- Max Bolingbroke has revised and simplified his Dynamically Loaded Plugins summer of code project, and we (continue to) plan to merge it into 6.12. Part of this is already merged: a new, modular system for annotations, rather like Java or C# attributes. These attributes are persisted into interface files, can be examined and created by plugins, or by GHC API clients.

- John Dias has continued work on rewriting GHC's backend. You can find an overview of the new architecture on the Wiki. He and Norman and Simon wrote Dataflow optimisation made simple, a paper about the dataflow optimization framework that the new back end embodies. Needless to say, the act of writing the paper has made us re-design the framework, so at the time of writing it still is not on GHC's main compilation path. But it will be.

- Shared Libraries are inching ever closer to being completed. Duncan Coutts has taken up the reins and is pushing our shared library support towards a fully working state. This project is supported by the Industrial Haskell Group.

- Unicode text I/O support is at the testing stage, and should be merged in in time for 6.12.1.

2.2 nhc98

Report by:	Malcolm Wallace
Status:	stable, maintained

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

Further reading

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

2.3 The Helium compiler

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

Helium is a compiler that supports a substantial subset of Haskell 98 (but, e.g., n+k patterns are missing). Type classes are restricted to a number of built-in type classes and all instances are derived. The advantage of

Helium is that it generates novice friendly error feedback. The latest versions of the Helium compiler are available for download from the new website located at http://www.cs.uu.nl/wiki/Helium. This website also explains in detail what Helium is about, what it offers, and what we plan to do in the near and far future.

We are still working on making version 1.7 available, mainly a matter of updating the documentation and testing the system. Internally little has changed, but the interface to the system has been standardized, and the functionality of the interpreters has been improved and made consistent. We have made new options available (such as those that govern where programs are logged to). The use of Helium from the interpreters is now governed by a configuration file, which makes the use of Helium from the interpreters quite transparent for the programmer. It is also possible to use different versions of Helium side by side (motivated by the development of Neon (→ 5.3.3)).

A student has added parsing and static checking for type class and instance definitions to the language, but type inferencing and code generating still need to be added. The work on the documentation has progressed quite a bit, but there has been little testing thus far, especially on a platform such as Windows.

2.4 UHC, Utrecht Haskell Compiler (previously: EHC, "Essential Haskell" compiler)

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

What is UHC? and EHC? UHC is the Utrecht Haskell Compiler, supporting almost all Haskell 98 features plus many experimental extensions. The first release of UHC was announced on April 18 at the 5th Haskell Hackathon, held in Utrecht.

EHC is the Essential Haskell Compiler project, a series of compilers of which the last is UHC, plus an aspectwise organized infrastructure for facilitating experimentation and extension.

The end-user will probably only be aware of UHC as a Haskell compiler, whereas compiler writers will be more aware of the internals known as EHC. The name EHC however will disappear over time, both EHC and UHC together will be branded as UHC.

UHC in its current state still very much is work in progress. Although we feel it is stable enough to offer the public, much work needs to be done to make it usable for serious development work. By design its strong point is the internal aspectwise organization which we started as EHC. UHC also offers more advanced and experimental features like higher-ranked polymorphism, partial type signatures, and local instances.

Under the hood For the description of UHC an Attribute Grammar system (AG) is used as well as other formalisms allowing compact notation like parser combinators. For the description of type rules, and the generation of an AG implementation for those type rules, we use the Ruler system. For source code management we use Shuffle, which allows partitioning the system into a sequence of steps and aspects. (Both Ruler and Shuffle are included in UHC).

The implementation of UHC also tackles other issues:

- To deal with the inherent complexity of a compiler the implementation of UHC is organized as a series of increasingly complex steps. Each step corresponds to a Haskell subset which itself is an extension of the previous step. The first step starts with the essentials, namely typed lambda calculus; the last step corresponds to UHC.
- Independent of each step the implementation is organized into a set of aspects. Currently the type system and code generation are defined as aspects, which can then be left out so the remaining part can be used as a barebones starting point.
- Each combination of step + aspects corresponds to an actual, that is, an executable compiler. Each of these compilers is a compiler in its own right.
- The description of the compiler uses code fragments which are retrieved from the source code of the compilers. In this way the description and source code are kept synchronized.

Part of the description of the series of EH compilers is available as a PhD thesis.

What is UHC's status, what is new?

- UHC has seen daylight as a first release. At the time of this writing the current release is 1.0.1, fixing reported installation problems.
- Previously started work is still continuing: GRIN backend, full program analysis (Jeroen Fokker), type system formalization and automatic generation from type rules (Lucília Camarão de Figueiredo, Arie Middelkoop).

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

- Improving installation of UHC and its use as a Haskell compiler: use of Cabal, adding missing Haskell 98 features.
- Work on adding static analyses (such as strictness analysis), to enable optimizations.

Further reading

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

2.5 Hugs as Yhc Core Producer

Report by:	Dimitry Golubovsky
Status:	experimental

Background

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

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

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

The experiment

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

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

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

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

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

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

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

Availability

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

Future plans

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

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

There were no significant changes, additions, or improvements made to the project since November 2008. After more testing and bugfixing, the Cabal package with Hugs front-end to Yhc Core was released on Hackage. Also, the `corehugs` project page was updated with new custom Hugs snapshot (as of February 16, 2009); this snapshot is recommended for experiments with Yhc Core.

Further reading

- Yhc Core conversion infrastructure http://tinyurl.com/ycore-conv-infra
- Download Hugs specialized snapshots http://code.google.com/p/corehugs/

- Proposed specification of the Normal Set of primitives
  http://tinyurl.com/prim-normal-set
- A brief example of using `corehugs`
  http://code.google.com/p/corehugs/wiki/Demonstration
- The hugs2yc package on Hackage
  http://hackage.haskell.org/cgi-bin/hackage-scripts/package/hugs2yc
- The `corehugs` project downloads page
  http://code.google.com/p/corehugs/downloads/list

2.6 Haskell frontend for the Clean compiler

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

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

Future plans

Although a beta version of the new Clean compiler is released last year to the institution in Nijmegen, there is still a lot of work to do before we are able to release it to the outside world. So we cannot make any promises regarding the release date. Just keep an eye on the Clean mailing lists for any important announcements!

Further reading

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

2.7 SAPL, Simple Application Programming Language

Report by:	Jan Martin Jansen
Status:	experimental, active development

SAPL is an experimental interpreter for a lazy functional intermediate language. The language is more or less equivalent to the core language of Clean (→ 3.2.3). SAPL implementations in C and Java exist. It is possible the write SAPL programs directly, but the preferred use is to generate SAPL. We already implemented an experimental version of the Clean compiler that generates SAPL as well. The Java version of the SAPL interpreter can be loaded as a PlugIn in web applications. Currently we use it to evaluate tasks from the iTask 6.8.3 system at the client side and to handle (mouse) events generated by a drawing canvas PlugIn.

Future plans

For the near future we have planned to make the Clean to SAPL compiler available in the standard Clean distribution. Also some further performance improvements of SAPL are planned.

Further reading

- http://home.hetnet.nl/~janmartinjansen/saplinter
- http://home.hetnet.nl/~janmartinjansen/lambda
- http://www.st.cs.ru.nl/Onderzoek/Publicaties/publicaties.html

2.8 The Reduceron

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

Since the last HCAR, we have started a 15-month project to further explore the potential of the Reduceron. I have begun implementation of a new design, and I presented initial results in talks at HFL'09 and at Birmingham University. Slides of both talks are available online. Once complete, we will have a rather nippy sequential reducer, and we plan to see if special-purpose architectural features can also aid garbage collection and parallel reduction.

There have also been some other developments in the project. I have made a new Lava (→ 6.9.2) clone supporting multi-output primitives, RAMs, easy creation of new primitives and back-ends, behavioral description, and sized-vectors. And Jason Reich is working on a supercompiler for the Reduceron's core language, exploring some of Neil Mitchell's ideas from Supero.

Further reading

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

2.9 Platforms

2.9.1 Haskell in Gentoo Linux

Report by:	Lennart Kolmodin

Gentoo Linux is working towards supporting GHC 6.10.3 and the Haskell Platform (→ 5.2), and at the

time of writing both are available hard masked in portage. For previous GHC versions we have binaries available for alpha, amd64, hppa, ia64, sparc, and x86.

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

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

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

There is also a Haskell overlay providing another 300 packages. Thanks to the haskell developers using Cabal and Hackage (→ 5.1), we have been able to write a tool called "hackport" (initiated by Henning Günther) to generate Gentoo packages that rarely need much tweaking.

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

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

2.9.2 Fedora Haskell SIG

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

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

A new cabal2spec package now generates rpm-spec files for Fedora's Haskell Packaging Guidelines from Cabal packages, making it easier than ever to package Haskell for Fedora. cabal2spec has been ported to Haskell by Konrad and Yaakov.

Fedora 11 will ship with ghc-6.10.2, cabal-install and an improved set of rpm macros at the end of May 2009.

For Fedora 12 we are planning to include xmonad and hopefully haskell-platform. The latest ghc.spec in development already supports building ghc-6.11 with shared libraries.

Further reading

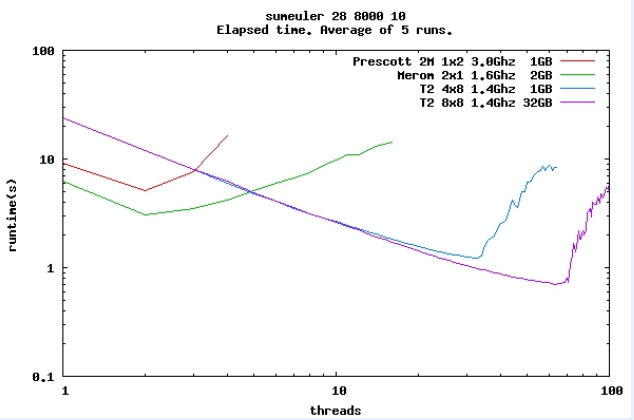
http://fedoraproject.org/wiki/SIGs/Haskell

2.9.3 GHC on OpenSPARC

Report by:	Ben Lippmeier
Participants:	Duncan Coutts, Darryl Gove, Roman Leshchinskiy
Status:	winding down

Through January–April this year I repaired GHC's back end support for the SPARC architecture, and benchmarked its performance on haskell.org's shiny new SPARC T2 server. I also spent time refactoring GHC's native code generator to make it easier to understand and maintain, and thus less likely for pieces to suffer bit-rot in the future.

The T2 architecture is interesting to functional programmers because of its highly multi-threaded nature. The T2 has eight cores with eight hardware threads each, for a total of 64 threads per processor. When one of the threads suffers a cache miss, another can continue on with little context switching overhead. All threads on a particular core also share the same L1 cache, which supports fast thread synchronization. This is a perfect fit for parallel lazy functional programs, where memory traffic is high, but new threads are only a `par` away. The following graph shows the performance of the sumeuler benchmark from the nofib suite when running on the T2. Note that the performance scales almost linearly (perfectly) right up to the point where it runs out of hardware threads.



The project is nearing completion, pending tying up some loose ends, but the port is fully functional and available in the current head branch. More information, including benchmarking is obtainable form the link below. The GHC on OpenSPARC project was generously funded by Sun Microsystems.

Further reading

http://ghcsparc.blogspot.com

3 Language

3.1 Extensions of Haskell

3.1.1 Haskell Server Pages (HSP)

Report by:	Niklas Broberg
Status:	active development

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

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

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

Further reading

http://haskell.org/haskellwiki/HSP

3.1.2 GpH — Glasgow Parallel Haskell

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

Status

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

System Evaluation and Enhancement

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

GpH Applications

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

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

Implementations

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

- The focus of the development has switched to versions tracking GHC releases, currently GHC 6.8, and the development version is available upon request to the GpH mailing list (see the GpH web site).
- The stable branch (GUM-4.06, based on GHC-4.06) is available for RedHat-based Linux machines. The stable branch is available from the GHC CVS repository via tag gum-4-06.

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

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

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

Further reading

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

Contact

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

3.1.3 Eden

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

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

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

Survey and standard reference

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

Implementation

A major revision of the parallel Eden runtime environment for GHC 6.8.1 is available from the Marburg group on request. Support for Glasgow parallel Haskell (→ 3.1.2) is currently being added to this version of the runtime environment. It is planned for the future to maintain a common parallel runtime environment for Eden, GpH, and other parallel Haskells. Program executions can be visualized using the Eden trace viewer tool EdenTV. Recent results show that the system behaves equally well on workstation clusters and on multi-core machines.

Recent and Forthcoming Publications

- Jost Berthold, Mischa Dieterle, Oleg Lobachev, Rita Loogen: *Distributed Memory Programming on Many-Cores - A Case Study Using Eden Divide-&-Conquer Skeletons*, in Workshop on Many-Cores at ARCS '09, Delft, NL, March 2009, VDE Verlag 2009.
- Mischa Dieterle, Jost Berthold, and Rita Loogen: *Implementing Parallel Google Map-Reduce in Eden*, in: EuroPar 2009, Delft, NL, August 2009, to appear in LNCS.
- Jost Berthold, Mischa Dieterle, Oleg Lobachev, Rita Loogen: *Parallel FFT With Eden Skeletons*, in PaCT 2009, Novosibirsk, Russia, September 2009, to appear in LNCS.
- Thomas Horstmeyer, Rita Loogen: *Grace — Graph-based Communication in Eden*, in Draft Proceedings of TFP, Kormarno, June 2009.
- Oleg Lobachev, Rita Loogen: *Benchmarking Parallel Haskells Using Scientific Computing Algorithms*, in Draft Proceedings of TFP, Kormarno, June 2009.
- Lidia Sánchez-Gil, Mercedes Hidalgo-Herrero, Yolanda Ortega-Mallén: *An Operational Semantics for Distributed Lazy Evaluation*, in Draft Proceedings of TFP, Kormarno, June 2009.

Further reading

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

3.1.4 XHaskell project

Report by:	Martin Sulzmann
Participants:	Kenny Zhuo Ming Lu

XHaskell is an extension of Haskell which combines parametric polymorphism, algebraic data types, and

type classes with XDuce style regular expression types, subtyping, and regular expression pattern matching. The latest version can be downloaded via http://code.google.com/p/xhaskell/

Latest developments

We are still in the process of turning XHaskell into a library (rather than stand-alone compiler). We expect results (a Hackage package) by August'09.

3.1.5 HaskellActor

Report by:	Martin Sulzmann

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

Novel features of HaskellActor include

- Multi-headed receive clauses, with support for
- guards, and
- propagation

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

The implementation is stable, but there is plenty of room for optimizations and extensions (e.g. regular expressions in patterns). If this sounds interesting to anybody (students!), please contact me.

Latest developments

We are currently collecting "real-world" examples where the HaskellActor extension (multi-headed receive clauses) proves to be useful.

3.1.6 HaskellJoin

Report by:	Martin Sulzmann

HaskellJoin is a (library) extension of Haskell to support join patterns. Novelties are

- guards and propagation in join patterns,
- efficient parallel execution model which exploits multiple processor cores.

Latest developments

Olivier Pernet (a student of Susan Eisenbach) is currently working on a nicer monadic interface to the HaskellJoin library.

Further reading

- http://sulzmann.blogspot.com/2008/12/parallel-join-patterns-with-guards-and.html
  The paper version is dated December'08. I will update the paper in the next couple of weeks.
- http://hackage.haskell.org/cgi-bin/hackage-scripts/package/join

3.2 Related Languages

3.2.1 Curry

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

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

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

Recently, new compilation schemes for translating Curry to Haskell have been developed that promise significant speedups compared to both the former KiCS implementation and other existing implementations of Curry.

There have been research activities in the area of functional logic programming languages for more than a decade. Nevertheless, there are still a lot of interesting research topics regarding more efficient compila-

tion techniques and even semantic questions in the area of language extensions like encapsulation and function patterns. Besides activities regarding the language itself, there is also an active development of tools concerning Curry (e.g., the documentation tool CurryDoc, the analysis environment CurryBrowser, the observation debuggers COOSy and iCODE, the debugger B.I.O. (http://www-ps.informatik.uni-kiel.de/currywiki/tools/oracle_debugger), EasyCheck (→ 4.3.2), and CyCoTest). Because Curry has a functional subset, these tools can canonically be transferred to the functional world.

Further reading

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

3.2.2 Agda

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

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

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

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

New since last time:

- Versions 2.2.0 and 2.2.2 have been released. The previous release was in 2007, so the new versions include lots of changes.
- Agda is now available on Hackage (`cabal install Agda-executable`).
- Highlighted, hyperlinked HTML can be generated from Agda source code using `agda -html`.
- The Agda Wiki is better organized, so it should be easier for a newcomer to find relevant information.

Further reading

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

3.2.3 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. Clean is the only functional language in the world which offers uniqueness typing. This type system makes it possible in a pure functional language to incorporate destructive updates of arbitrary data structures (including arrays) and to make direct interfaces to the outside imperative world.

Here is a short list with notable features:

- Clean is a lazy, pure, 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.
- The uniqueness type 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. The uniqueness type system can be also used to incorporate destructive updates of objects within a pure functional framework. It allows destructive transformation of state information and enables efficient interfacing to the non-functional world (to C but also to I/O systems like X-Windows) offering direct access to file systems and operating systems.
- Clean supports type classes and type constructor classes to make overloaded use of functions and operators possible.
- Clean offers records and (destructively updateable) arrays and files.
- Clean has pattern matching, guards, list comprehensions, array comprehensions and a lay-out sensitive mode.
- Clean offers a sophisticated I/O library with which window based interactive applications (and the handling of menus, dialogs, windows, mouse, keyboard,

timers, and events raised by sub-applications) can be specified compactly and elegantly on a very high level of abstraction.

- There is a Clean IDE and there are many libraries available offering additional functionality.

Future plans

Please see the entry on a Haskell frontend for the Clean compiler (→ 2.6) for the future plans.

Further reading

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

3.2.4 Timber

Report by:	Johan Nordlander
Participants:	Björn von Sydow, Andy Gill, Magnus Carlsson, Per Lindgren, 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.

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 first public release of the Timber compiler was announced in December 2008. It uses the Gnu C compiler as its back-end and targets POSIX-based operating systems. Binary installers for Linux and MacOS X can be downloaded from the Timber web site timber-lang.org.

The current source code repository (also available on-line) contains numerous bug-fixes since the release, but also initial cross-compilation support for ARM7-equipped embedded systems. A proper release of the current version is in preparation.

Apart from this release, on-going work on Timber is concerned with extending the compiler with a supercompilation pass, adding the iPhone as a cross-compilation target, and taking a fundamental grip on the generation of type error messages. The latter work will be based on the principles developed in the context of the Helium compiler.

Further reading

http::/timber-lang.org

3.3 Type System / Program Analysis

3.3.1 Free Theorems for Haskell

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

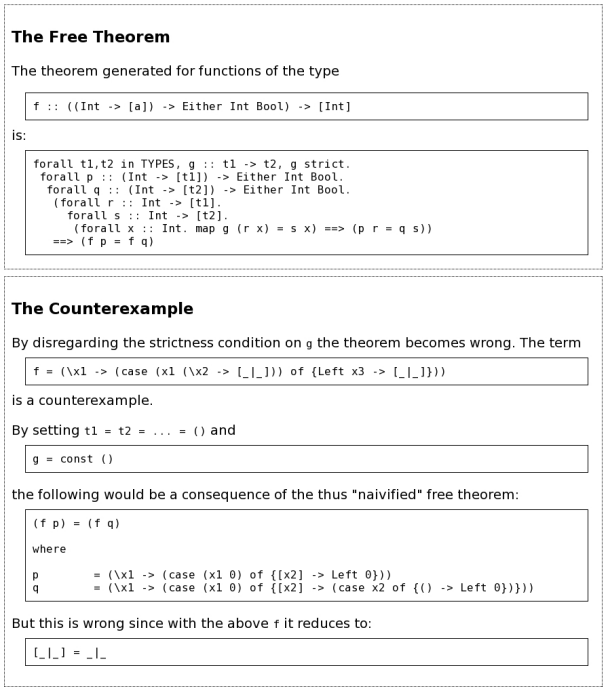
Free theorems are statements about program behavior derived from (polymorphic) types. Their origin is the polymorphic lambda-calculus, but they have also been applied to programs in more realistic languages like Haskell. Since there is a semantic gap between the original calculus and modern functional 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. Recent publications are "Parametricity for Haskell with Imprecise Error Semantics" (TLCA'09) and "Free Theorems Involving Type Constructor Classes" (ICFP'09).

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. Both the library and a shell-based tool are available from Hackage (as free-theorems and ftshell, respectively). There is also a web-based tool at http://linux.tcs.inf.tu-dresden.de/~voigt/ft. General features include:

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

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

Recent work with Daniel Seidel involves automatically generating counterexamples to naive free theorems ("naive" meaning that the theorems ignore the presence of general recursion and $\perp$ in Haskell; see screenshot below), and investigating how selective strictness á la `seq` has to be put under control to tame its impact on free theorems.



The Free Theorem

The theorem generated for functions of the type

```
f :: ((Int -> [a]) -> Either Int Bool) -> [Int]
```

is:

```
forall t1,t2 in TYPES, g :: t1 -> t2, g strict.
 forall p :: (Int -> [t1]) -> Either Int Bool.
  forall q :: (Int -> [t2]) -> Either Int Bool.
   (forall r :: Int -> [t1].
      forall s :: Int -> [t2].
        (forall x :: Int. map g (r x) = s x) ==> (p r = q s))
  ==> (f p = f q)
```

The Counterexample

By disregarding the strictness condition on `g` the theorem becomes wrong. The term

```
f = (\x1 -> (case (x1 (\x2 -> [_|_])) of {Left x3 -> [_|_]}))
```

is a counterexample.

By setting `t1 = t2 = ... = ()` and

```
g = const ()
```

the following would be a consequence of the thus "naivified" free theorem:

```
(f p) = (f q)

where

p       = (\x1 -> (case (x1 0) of {[x2] -> Left 0}))
q       = (\x1 -> (case (x1 0) of {[x2] -> (case x2 of {() -> Left 0})}))
```

But this is wrong since with the above `f` it reduces to:

```
[_|_] = _|_
```

Further reading

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

3.3.2 The Disciplined Disciple Compiler (DDC)

Report by:	Ben Lippmeier
Status:	alpha, active

Disciple is an explicitly lazy dialect of Haskell which is being developed as part of my PhD project into program optimization in the presence of side effects. Disciple's type system is based on Haskell 98, but extends it with region, effect and closure information. This extra information models the potential aliasing of data, interference of computations, and the data sharing properties of functions.

Effect typing is offered as a practical alternative to source level state monads, and allows the programmer to offload the task of maintaining the intended sequence of computations onto the compiler. Disciple can also directly express the "flat" T-monads discussed in Wadler's "The marriage of effects and monads", if one were that way inclined.

Disciple uses strict evaluation as the default, but the programmer can suspend arbitrary function applications if desired. As the type system detects when the combination of side effects and call-by-need evaluation would yield a result different from the call-by-value case, we argue that Disciple is still a purely functional language by Sabry's definition.

Future plans

DDC is currently a research prototype, but will compile programs if you are nice to it. Work over the last six months has consisted of cleaning up the theory, finishing my thesis, and fixing bugs. Immediate plans consist of fixing more bugs, doing a point release in July, and writing a paper on it. DDC is open source, available from the url below, and comes with some cute graphical demos.

Further reading

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

4 Tools

4.1 Scanning, Parsing, Transformations

4.1.1 Alex version 2

Report by:	Simon Marlow
Status:	stable, maintained

Alex is a lexical analyzer generator for Haskell, similar to the tool lex for C. Alex takes a specification of a lexical syntax written in terms of regular expressions, and emits code in Haskell to parse that syntax. A lexical analyzer generator is often used in conjunction with a parser generator, such as Happy (→ 4.1.2), to build a complete parser.

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

Changes in version 2.3 vs. 2.2:

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

Further reading

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

4.1.2 Happy

Report by:	Simon Marlow
Status:	stable, maintained

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

The latest release is 1.18.2, released 5 November 2008.

Changes in version 1.18.2 vs. 1.17:

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

Further reading

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

4.1.3 UUAG

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

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

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

The system is in use by a variety of large and small projects, such as the Haskell compiler EHC, the editor Proxima for structured documents, the Helium compiler (→ 2.3), the Generic Haskell compiler, and UUAG itself. The current version is 0.9.10 (April 2009), is extensively tested, and is available on Hackage.

The last year, there have been bugfix releases only, and one feature has been introduced. An alternative syntax is now available which resembles the Haskell syntax more closely. This syntax can be enabled by means of a commandline flag. The old syntax is still the default.

Further reading

- http://www.cs.uu.nl/wiki/bin/view/HUT/AttributeGrammarSystem
- http://hackage.haskell.org/packages/archive/uuagc/0.9.10/uuagc-0.9.10.tar.gz

4.2 Documentation

4.2.1 Haddock

Report by:	David Waern
Status:	experimental, maintained

Haddock is a widely used documentation-generation tool for Haskell library code. Haddock generates documentation by parsing the Haskell source code directly and including documentation supplied by the programmer in the form of specially-formatted comments in the source code itself. Haddock has direct support in Cabal (→ 5.1), and is used to generate the documentation for the hierarchical libraries that come with GHC, Hugs, and nhc98 (http://www.haskell.org/ghc/docs/latest/html/libraries).

The latest release is version 2.2.2, released August 5 2008.

Recent changes:

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

Future plans

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

Other things we plan to add in future releases:

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

Further reading

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

4.2.2 lhs2TEX

Report by:	Andres Löh
Status:	stable, maintained

This tool by Ralf Hinze and Andres Löh is a preprocessor that transforms literate Haskell code into LaTeX documents. The output is highly customizable by means of formatting directives that are interpreted by lhs2TeX. Other directives allow the selective inclusion of program fragments, so that multiple versions of a program and/or document can be produced from a common source. The input is parsed using a liberal parser that can interpret many languages with a Haskell-like syntax, and does not restrict the user to Haskell 98.

The program is stable and can take on large documents.

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

Further reading

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

4.3 Testing, Debugging, and Analysis

4.3.1 SmallCheck and Lazy SmallCheck

Report by:	Matthew Naylor
Participants:	Fredrik Lindblad, Colin Runciman
Status:	stable, maintained

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

Lazy SmallCheck is like SmallCheck, but generates partially-defined inputs that are progressively refined as demanded by the property under test. The key observation is that if a property evaluates to True or False for a partially-defined input then it would also

do so for all refinements of that input. By not generating such refinements, Lazy SmallCheck may test the same input-space as SmallCheck using significantly fewer tests. Lazy SmallCheck's interface is a subset of SmallCheck's, often allowing the two to be used interchangeably.

Since the last HCAR, we have not made any significant new developments. We are still interested in improving and harmonizing the two libraries and welcome comments and suggestions from users.

Further reading

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

4.3.2 EasyCheck

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

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

Further reading

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

4.3.3 checkers

Report by:	Conal Elliott
Status:	active development

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

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

Further reading

http://haskell.org/haskellwiki/checkers

4.3.4 Gast

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

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

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

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

Future plans

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

Further reading

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

4.3.5 Concurrent Haskell Debugger

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

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

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

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

Future plans

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

Further reading

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

4.3.6 Hpc

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

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

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

Contact

⟨andygill@ku.edu⟩

4.3.7 SourceGraph

Report by:	Ivan Lazar Miljenovic
Status:	version 0.3

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

Whilst fully usable, SourceGraph is currently limited in terms of input and output. It takes in the Cabal file of the project, and then analyzes all .hs and .lhs files recursively found in that directory. It utilizes HaskellSrc with Extensions, and should thus parse all extensions (with the current exceptions of Template Haskell, HaRP, and HSX); files requiring C Pre-Processing are as yet unparseable, though this should be fixed in a future release. However, all functions defined in Class declarations and records are ignored due to difficulty in determining which actual instance is meant. The final report is then created in Html format in a "SourceGraph" subdirectory of the project's root directory. It is envisaged that future versions will at least allow the user to choose which output format to produce the report in, and even customize which analyses are performed (e.g., just create the graph of the entire codebase, and not perform any analysis).

Current analysis algorithms utilized include: alternative module groupings, whether a module should be split up, root analysis, clique and cycle detection as well as finding functions which can safely be compressed down to a single function. Please note however that SourceGraph is *not* a refactoring utility, and that its

analyses should be taken with a grain of salt: for example, it might recommend that you split up a module, because there are several distinct groupings of functions, when that module contains common utility functions that are placed together to form a library module (e.g., the Prelude).

The output from running SourceGraph on itself can be found at http://community.haskell.org/~ivanm/SourceGraph/SourceGraph.html.

Further reading

- http://hackage.haskell.org/cgi-bin/hackage-scripts/package/SourceGraph
- http://code.haskell.org/SourceGraph
- http://ivanmiljenovic.files.wordpress.com/2008/11/honoursthesis.pdf

4.3.8 HLint

Report by:	Neil Mitchell
Status:	v1.4

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.

Further reading

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

4.3.9 hp2any

Report by:	Patai Gergely
Status:	planned

`hp2any` is the codename for the 2009 Google Summer of Code project titled "Improving space profiling experience".

The aim of the project is to create a set of tools that make heap profiling of Haskell programs easier. In particular, the following components are planned:

- a library to process profiler output in an efficient way and make it easily accessible for other tools;
- a real-time visualizer (most likely using OpenGL);
- some kind of history manager to keep track of profiling data and make it possible to perform a comparative analysis of performance between different versions of your program;
- a maintainable and extensible replacement for hp2ps.

Community input is greatly appreciated!

Further reading

- http://code.google.com/p/hp2any/
- http://socghop.appspot.com/student_project/show/google/gsoc2009/haskell/t124022468245

4.4 Development

4.4.1 Hoogle — Haskell API Search

Report by:	Neil Mitchell
Status:	v4.0

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

Work is progressing to generate Hoogle search information for all the libraries on Hackage.

Further reading

http://haskell.org/hoogle

4.4.2 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.

The most recent Version 3.1 fixes some bugs and makes Heat work more smoothly on Macs.

Further reading

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

4.4.3 Haskell Mode Plugins for Vim

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

The Haskell mode plugins for Vim offer IDE-style programmer assistance for editing Haskell code in Vim. Functionality and semantic information are derived from GHC/GHCi, from Haddock-generated documentation and indices (→ 4.2.1), and from instantiating Vim's own configurable program editing support with Haskell-specific information. The plugins are distributed as a simple vimball archive, including help file (after installation: `:help haskellmode`), and should work on most platforms (Windows, Mac, Unix), in terminal or GUI mode (some configuration required).

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

The haskellmode plugins for Vim are currently in maintenance mode, with infrequent updates and bug fixes, and the occasional new feature or improvement of older code (please let me know if anything does not work as advertised!). They have just moved to haskell.org, where they have acquired a trac instance. Apart from various bug- and portability fixes, recent additions include a short series of screencasts illustrating most features, as well as Hoogle and Hayoo! lookup for those who edit their programs while being online.

Further reading

- Haskell Mode Plugins for Vim: http://projects.haskell.org/haskellmode-vim/
- Screencasts, Documentation, Change Log:
  http://projects.haskell.org/haskellmode-vim/screencasts.html
  http://projects.haskell.org/haskellmode-vim/vimfiles/doc/haskellmode.txt
  http://projects.haskell.org/haskellmode-vim/vimfiles/haskellmode-files.txt
- haskell.org section listing these and other Vim files (please add your own): http://www.haskell.org/haskellwiki/Libraries_and_tools/Program_development#Vim

4.4.4 yi

Report by:	Jean-Philippe Bernardy
Participants:	Nicolas Pouillard, Jeff Wheeler, and many others
Status:	active development

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

Defining characteristics:

- A purely functional buffer representation;
- Powerful EDSLs to describe editor actions and keybindings;
- Syntax-highlighters as Alex files;
- XMonad-style static/dynamic configuration;
- UIs written as plugins.

We are currently working on making the Gtk frontend fully usable and improving the syntax highlighters. A lot of work is also put into making the transition from emacs or vim smoother.

Features:

- Special support for Haskell: layout-aware edition, paren-matching, beautification of lambdas and arrows, GHCi interface, Cabal build interface, ...
- unix console UI;
- Support for Linux and MacOS platforms;
- Syntax highlighting for many mainstream languages beside Haskell;

Further reading

- More information can be found at: http://haskell.org/haskellwiki/Yi

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

4.4.5 HaRe — The Haskell Refactorer

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

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

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

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

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

Chris Brown has passed his PhD viva; his PhD thesis entitled "Tool Support for Refactoring Haskell Programs" will be available from our webpage soon.

Recent developments

- More structural and datatype-based refactorings have been studied by Chris Brown, including transformation between `let` and `where`, generative folding, introducing pattern matching, and introducing case expressions;
- Clone detection and elimination support has been added, to allow the automatic detection and semi-automatic elimination of duplicated code in Haskell.
- Chaddaï Fouché started to work on the porting of HaRe to GHC API (→ 2.1) during the summer 2008, and this work is ongoing.

Further reading

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

4.4.6 DarcsWatch

Report by:	Joachim Breitner
Status:	working

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

During the last six months, no changes to DarcsWatch were made, with the exception of some JavaScript to load the toggleable display of the changes of a patch on demand. It continues to be used by the xmonad project (→ 6.1.2) and a few developers. At the time of writing, it was tracking 26 repositories and 2252 patches submitted by 129 users.

Further reading

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

4.4.7 cpphs

Report by:	Malcolm Wallace
Status:	stable, maintained

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

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

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

Further reading

http://haskell.org/cpphs

5 Libraries

5.1 Cabal and Hackage

Report by: Duncan Coutts

Background

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

Hackage is 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

There has been some, but not a huge amount of activity since the last HCAR. There have been a couple of minor releases in the Cabal-1.6 series and a release of cabal-install which is now at version 0.6.2. These releases addressed a number of issues that were plaguing users.

The cabal-install tool is now relatively mature (though by no means perfect). It replaces `runhaskell Setup.hs` which had been the primary interface for most users previously. The major advantage is that it simplifies the process of downloading and installing collections of inter-dependent packages from Hackage. The primary problem with cabal-install now is getting it into the hands of end users. This is being addressed by the Haskell Platform (→ 5.2) initiative.

Hackage passed the 1000 package mark, in fact the number of packages since the last HCAR has increased by about 50%. This represents a substantial amount of Haskell code and a substantial amount of code re-use.

Looking forward

As ever, there are many improvements we want to make to Cabal, cabal-install and Hackage but our limiting factor is the amount of volunteer development time. We have nearly 50 tickets targeted for Cabal 1.8 but very few are currently being worked on. In addition to the immediate task of refactoring, fixing bugs and adding new features, the next big challenge is improving Cabal to handle larger projects which have more complex requirements for a configuration and build system. This challenge may require us to take a step back and discuss a new design document for Cabal 2.0. Now would be an excellent time to get involved in this central piece of the Haskell infrastructure.

Hackage has a huge potential to help us manage and improve the community's package collection. cabal-install is now able to report build results and the new Hackage server implementation can accept them. This should provide us with a huge amount of data on which packages work in which environments and configurations. More generally there is the opportunity to collect all sorts of useful metrics on the quality of packages. The new Hackage server implementation exists but needs more work before it reaches feature parity with the current implementation. This is another important project that needs more developer time.

Further reading

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

5.2 Haskell Platform

Report by: Duncan Coutts

Background

The Haskell Platform (HP) is the name of a new "blessed" set of libraries and tools on which to build further Haskell libraries and applications. It takes the best packages from the more than 1000 on Hackage (→ 5.1). It is intended to provide a comprehensive, stable, and quality tested base for Haskell projects to work from.

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

Recent progress

By the time you read this we will have had the first beta release of the platform, version 2009.2.0. This first release contains the packages from the old `extralibs` collection, plus `cabal-install` (→ 5.1).

There will be follow-up minor releases 4 weeks and 10 weeks after the initial release. These will incorporate feedback on the installers and packaging. Your comments and feedback will be appreciated.

The intention of this first major release series is to get up to speed and test out our systems for making releases. We want to have everything working smoothly in time for GHC 6.12 when we hope to take over from the GHC team the task of making end-user releases.

Now is therefore the time to make your suggestions or to get involved with practical contributions towards automating the process of making quality releases.

Looking forward

Major releases will be on a 6 month schedule. Major releases may include new and updated packages while minor releases will contain bug fixes and fixes for packaging problems.

This is a project that must be owned by the community. We will be relying on the community to agree policies such as what procedures we should use and what level of quality we should demand for additions to the platform. The discussion will take place on the `libraries@haskell.org` mailing list, so subscribe if you wish to have your say. We will also be looking for members of a steering committee to guide the discussions on the libraries mailing list to ensure that the necessary decisions do actually get made, recorded and communicated to the release engineering team.

Further reading

http://haskell.org/haskellwiki/Haskell_Platform

5.3 Auxiliary Libraries

5.3.1 libmpd

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

LIBMPD is a client implementation of the MPD music playing daemon's network protocol. The interface has mostly stabilized and is usable. Although the next release will have an improved API using a typeclass, it should be source compatible. In version 0.3.1 some bugs have been addressed to fix the automatic reconnection feature and to be more permissive with data from the server.

Support for bytestrings is planned for the future.

Further reading

The project's web page is at http://projects.haskell.org/libmpd/ and MPD can be found at http://www.musicpd.org/.

5.3.2 hmatrix

Report by:	Alberto Ruiz
Status:	stable, maintained

This is a purely functional library for numerical linear algebra, internally implemented using GSL, BLAS, and LAPACK. The latest stable version is available from Hackage.

Future plans include support for multidimensional arrays, and a possible division of the library into smaller, independent packages.

Further reading

http://www.hmatrix.googlepages.com

5.3.3 The Neon Library

Report by:	Jurriaan Hage

As part of his master thesis work, Peter van Keeken implemented a library to data mine logged Helium (→ 2.3) 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.

The recent news is that a paper about Neon will be published at SLE (1st Conference on Software Language Engineering), where it came under the heading of Tools for Language Usage.

On that note, there has been a posting by Simon Thompson, Sally Fincher and myself for a PhD student to work on understanding how students learn to program (in Haskell), in Kent. Also, recently I acquired a new master student to continue to the work of Peter van Keeken. One of this tasks will be to investigate the kind of parse errors students make, and continue to make. In the process, he shall add context properties (did the student pass or fail, what kind of programming background can we expect him or her to have) to our database so that they can be employed by queries to increase external validity.

5.3.4 unamb

Report by:	Conal Elliott
Status:	active development

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

Further reading

http://haskell.org/haskellwiki/unamb

5.3.5 leapseconds-announced

Report by:	Björn Buckwalter
Status:	stable, maintained

The leapseconds-announced library provides an easy to use static LeapSecondTable with the leap seconds announced at library release time. It is intended as a quick-and-dirty leap second solution for one-off analyses concerned only with the past and present (i.e. up until the next as of yet unannounced leap second), or for applications which can afford to be recompiled against an updated library as often as every six months.

Version 2009 of leapseconds-announced contains all leap seconds up to 2009-01-01. A new version will be uploaded if/when the IERS announces a new leap second.

Further reading

- http://hackage.haskell.org/cgi-bin/hackage-scripts/package/leapseconds-announced
- http://github.com/bjornbm/leapseconds-announced

5.4 Processing Haskell

5.4.1 hint

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

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

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

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

Further reading

The latest stable version can be downloaded from Hackage.

5.4.2 mueval

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

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

Mueval features:

- A comprehensive test-suite of expressions which should and should not work
- Defeats all known attacks
- Optional resource limits and module imports
- The ability to load in definitions from a specified file
- Parses Haskell expressions with haskell-src-exts and tests against black- and white-lists
- A process-level watchdog, to work around past and future GHC issues with thread-level watchdogs
- Cabalized

We are currently working on the following:

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

Further reading

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

5.4.3 hscolour

Report by:	Malcolm Wallace
Status:	stable, maintained

HsColour is a small command-line tool (and Haskell library) that syntax-colorizes Haskell source code for multiple output formats. It consists of a token lexer, classification engine, and multiple separate pretty-printers for the different formats. Current supported output formats are ANSI terminal codes, HTML (with or without CSS), LaTeX, and IRC chat codes. In all cases, the colors and highlight styles (bold, underline, etc.) are configurable. It can additionally place HTML anchors in front of declarations, to be used as the target of links you generate in Haddock (→ 4.2.1) documentation.

HsColour is widely used to make source code in blog entries look more pretty, to generate library documentation on the web, and to improve the readability of GHC's intermediate-code debugging output. The current version is 1.10, which simply improves the title element on HTML output.

Further reading

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

5.5 Parsing and Transforming

5.5.1 HStringTemplate

Report by:	Sterling Clover

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

Further reading

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

5.5.2 CoreErlang

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

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

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

Further reading

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

5.5.3 parse-dimacs: A DIMACS CNF Parser

Report by:	Denis Bueno
Status:	version 1.2

Parse-dimacs parses a conjunctive normal form (CNF) constraints in the DIMACS file format. CNF formulas are often used as input to satisfiability solvers. DIMACS is a typical format for such formulas.

Version 1.2 is implemented in terms of lazy bytestrings, which provides some speedups over the previous string-based implementation.

Further reading

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

5.5.4 InterpreterLib

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

The InterpreterLib library is a collection of modules for constructing composable, monadic interpreters in Haskell. The library provides a collection of functions and type classes that implement semantic algebras in the style of Hutton and Duponcheel. Datatypes for related language constructs are defined as functors and composed using a higher-order sum functor. The full AST for a language is the least fixed point of the sum

of its constructs' functors. To denote a term in the language, a sum algebra combinator composes algebras for each construct functor into a semantic algebra suitable for the full language, and the catamorphism introduces recursion. Another piece of InterpreterLib is a novel suite of algebra combinators conducive to monadic encapsulation and semantic re-use. The library also implements a specialization of the SmashA generic programming technique to support generic default algebras and to override those defaults with functor-specific behavior. The Algebra Compiler, an ancillary preprocessor derived from polytypic programming principles, generates functorial boilerplate Haskell code from minimal specifications of language constructs. As a whole, the InterpreterLib library enables rapid prototyping, re-use, and simplified maintenance of language processors.

The Oread (→ 6.9.4) implementation employs InterpreterLib.

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

Further reading

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

Contact

⟨nfrisby@ittc.ku.edu⟩

5.5.5 KURE

Report by:	Andy Gill
Status:	alpha

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

Contact

⟨andygill@ku.edu⟩

5.5.6 Typed Transformations of Typed Abstract Syntax (TTTAS)

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

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

Introduction

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

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

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

Background

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

The library is employed to implement the transformations used in the Haskell 2008 paper "Haskell, Do You Read Me? Constructing and Composing Efficient Top-down Parsers at Runtime" (→ 5.5.7).

Future plans

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

Further reading

More information can be found on the TTTAS home page.

5.5.7 ChristmasTree (previously: GRead)

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

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

Introduction

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

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

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

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

Background

The approach taken in ChristmasTree was proposed by Marcos Viera, Doaitse Swierstra, and Eelco Lempsink in the Haskell 2008 paper "Haskell, Do You Read Me? Constructing and Composing Efficient Top-down Parsers at Runtime." It uses the Typed Transformations of Typed Abstract Syntax library (→ 5.5.6) developed by Arthur Baars and Doaitse Swierstra.

Further reading

- In a GHC bug ticket the problem we solved with this library is explained.
- More information can be found on the TTTAS home page.

5.5.8 Utrecht Parser Combinator Library: Old version

Report by:	Doaitse Swierstra
Status:	actively maintained

The old version of the Utrecht Parser Combinator library (part of the *uulib* package) has remained largely unmodified. One of its main uses is in the recently released Utrecht Haskell Compiler. In the course of testing that compiler versus the standard GHC packages we ran into a small misinterpretation of the offside rule, which has been repaired.

A more serious problem is that the GHC actually accepts a wider class of programs than specified by the offside rule as specified in the Haskell 98 standard. We may have to change out standard *pBlock* combinator in order to cope with this. The solution for the time being is to add some extra indentation.

Features

Fast, online, no spaceleaking implementation; stable applicative style interface.

Contact

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

5.5.9 Utrecht Parser Combinator Library: New version

Report by:	Doaitse Swierstra
Status:	actively developed

The Utrecht Parser Combinator library has remained largely unmodified for the last five years, and has served us well. Over the years, however, new insights have grown, and with the advent of GADTs some internals could be simplified considerably. The Lernet summer school in February 2008 (http://www.fing.edu.uy/inco/eventos/lernet2008/) provided an incentive to start a rewrite of the library; a newly written tutorial will appear in the lecture notes, which will be published by Springer in the LNCS series. The text is also available as a technical report at http://www.cs.uu.nl/research/techreps/UU-CS-2008-044.html

Features

- Much simpler internals than the old library (→ 5.5.8).
- Online result production, error recovery, combinators for parsing ambiguous grammars, an applicative interface, a monadic interface.
- Scanners can be switched dynamically, so several different languages can occur intertwined in a single input file.
- Fixes a potential black hole which went unnoticed for years in the code for the monadic bind as presented by Swierstra and Hughes in the ICFP 2003 paper: *Polish Parsers: Step by Step*.

A first version of the new library was recently released as the *uu-parsinglib* library, which has found its place in the *Text.ParserCombinators* category on Hackage.

Future plans

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

Since many aspects of the old library, such as its applicative interface and the possibility to build e.g. parser for permutation phrases, have now come available elsewhere in other packages, we will also try to make the new library to conform as much as possible with these new developments.

Contact

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

5.6 Mathematical Objects

5.6.1 Halculon: units and physical constants database

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

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

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

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

Further reading

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

5.6.2 Numeric prelude

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

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

There are both certain new type classes representing algebraic structures and new types of mathematical objects. Currently supported algebraic structures are group (additive), ring, principal ideal domain, field, algebraic closures, transcendental closures, module and vector space, normed space, lattice, differential algebra, monoid.

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

mial (basics only), Gaussians for exact Fourier transform, permutation group.

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

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

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

Further reading

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

5.6.3 vector-space

Report by:	Conal Elliott
Status:	active development

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

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

Further reading

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

5.6.4 Nat

Report by:	Jan Christiansen
Status:	experimental

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

Further reading

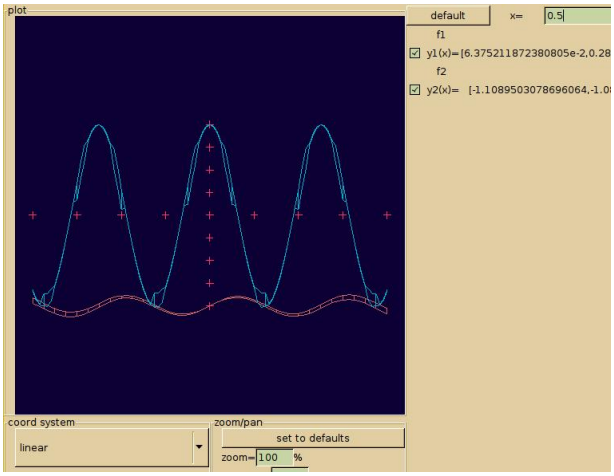
http://www-ps.informatik.uni-kiel.de/currywiki/fun/naturals

5.6.5 AERN-Real and friends

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

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

- AERN-Real: arbitrary precision interval arithmetic with multiple backends (pure Haskell floating point numbers, MPFR, correctly rounded doubles)
- AERN-RnToRm: arbitrary precision arithmetic of piece-wise polynomial function enclosures (PFEs) for functions over n-dimensional real intervals
- AERN-RnToRm-Plot: GTK window for inspecting the graphs of PFEs in one variable (see figure below, showing a screenshot of an AERN-RnToRm-Plot window exploring an enclosure of $cos(10x)$ (blue) and an enclosure of its primitive function (red))
- AERN-Net: an implementation of distributed query-based (i.e., lazy) computation over analytical and geometrical objects



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

- prototype algorithms for reliable and ultimately converging methods for solving differential equations in many variables (AERN-RnToRm, AERN-Net)
- solve numerical constraint satisfaction problems, especially those arising from verification of programs that use floating point numbers (AERN-RnToRm)

For our purposes AERN-Real has been stable for almost a year and a half. It needs to be tested for a wider variety of applications before we can label it as stable. AERN-RnToRm is now also fairly stable thanks to a period of debugging and a comprehensive test suite. Nevertheless, it is rather slow as it has not been optimized yet. The other libraries are very likely to contain errors and we discover some every now and then.

The API of all the libraries is still occasionally changing but they provide a fairly extensive set of features and are reasonably well documented.

The libraries are under active development and new features and bug fixes are expected to be submitted to Hackage during the rest of 2009. Notable planned additions in this period include:

- optimizations to the function enclosure arithmetic
- inner-rounded interval and function enclosure operations suitable for verification of interval inclusions
- infinite trees of enclosures for interval partial derivatives computed using automatic differentiation
- zooming, panning and better coordinate display in the GTK graph display

Further reading

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

5.6.6 Haskell BLAS Bindings

Report by:	Patrick O. Perry
Status:	version 0.6

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

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

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

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

Further reading

http://quantile95.com

5.6.7 logfloat

Report by:	Wren Ng Thornton
Status:	stable?
Current release:	0.12.0.1
Portability:	GHC 6.8, GHC 6.10, Hugs Sept2006

The logfloat library provides a type for storing numbers in the log-domain. This is primarily useful for avoiding underflow when multiplying many small numbers in probabilistic models.

It also includes support for dealing with IEEE-754 floating point numbers (more) correctly, including: a class for types with representations for transfinite values, a class for partially ordered types, efficient and correct conversion from `Real` to `Fractional`, and bug fixes for Hugs' Prelude.

Future plans

Add a signed variant so negative numbers can also be projected into the log-domain.

Further reading

- Official source and documentation available on Hackage
- The development branch is available from http://community.haskell.org/~wren/

5.6.8 fad: Forward Automatic Differentiation

Report by:	Björn Buckwalter
Participants:	Barak A. Pearlmutter, Jeffrey Mark Siskind
Status:	active

Fad is an attempt to make as comprehensive and usable a forward automatic differentiation (AD) library as is possible in Haskell. Fad (a) attempts to be correct, by making it difficult to accidentally get a numerically incorrect derivative; (b) provides not only first-derivatives, but also a lazy tower of higher-order derivatives; (c) allows nested use of derivative operators while using the type system to reject incorrect nesting (perturbation confusion); (d) attempts to be complete, in the sense of allowing calculation of derivatives of functions defined using a large variety of Haskell constructs; and (e) tries to be efficient, in the sense of both

the defining properties of forward automatic differentiation and in keeping the constant factor overhead as low as possible.

Version 1.0 of fad was uploaded to Hackage on April 3. Recent changes can be found via `git clone git://github.com/bjornbm/fad.git`

Further reading

- http://github.com/bjornbm/fad
- http://flygdynamikern.blogspot.com/2009/04/announce-fad-10-forward-automatic.html

5.7 Data types and data structures

5.7.1 HList — a library for typed heterogeneous collections

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

HList is a comprehensive, general purpose Haskell library for typed heterogeneous collections including extensible polymorphic records and variants (→ 1.5). HList is analogous to the standard list library, providing a host of various construction, look-up, filtering, and iteration primitives. In contrast to the regular lists, elements of heterogeneous lists do not have to have the same type. HList lets the user formulate statically checkable constraints: for example, no two elements of a collection may have the same type (so the elements can be unambiguously indexed by their type).

An immediate application of HLists is the implementation of open, extensible records with first-class, reusable, and compile-time only labels. The dual application is extensible polymorphic variants (open unions). HList contains several implementations of open records, including records as sequences of field values, where the type of each field is annotated with its phantom label. We, and now others (Alexandra Silva, Joost Visser: PURe.CoddFish project), have also used HList for type-safe database access in Haskell. HList-based Records form the basis of OOHaskell (http://darcs.haskell.org/OOHaskell). The HList library relies on common extensions of Haskell 98.

We are presently re-integrating the regression test suite into the Hackage HList distribution and adjusting the library for GHC 6.10.x. We are investigating the use of type functions provided in the new versions of GHC.

Further reading

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

5.7.2 Edison

Report by:	Robert Dockins
Status:	stable, maintained

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

1. A set of type classes defining data the following data structure abstractions: “sequences”, “collections” and “associative collections”
2. Multiple concrete implementations of each of the abstractions.

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

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

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

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

Further reading

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

5.7.3 MemoTrie

Report by:	Conal Elliott
Status:	active development

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

Further reading

http://haskell.org/haskellwiki/MemoTrie

5.7.4 bytestring-trie

Report by:	Wren Ng Thornton
Status:	active development
Current release:	0.1.4
Portability:	Haskell 98 + CPP

The bytestring-trie library provides an efficient implementation of "dictionaries" mapping strings to values, using big-endian patricia tries (like `Data.IntMap`). In general `Trie` is more efficient than `Map ByteString` because memory and work is shared between strings with common prefixes, though the specifics will vary depending on the distribution of keys.

Future plans

- Min- and max-views for treating tries as priority queues.
- Efficient intersection and difference functions.

Further reading

- Official source and documentation available on Hackage.
- The development branch is available from http://community.haskell.org/~wren/.

5.8 Data processing

5.8.1 The Haskell Cryptographic Library

Report by:	Creighton Hogg

The latest version is 4.2.0. It is primarily a bug fix over 4.1.0 that should be cabal installable by any recent GHC.

Having taken over the project from Dominic, the primary focus for the 5.0.0 release of Crypto will be improved speed and bringing the entire codebase under a BSD license.

Please note that the project has moved over to the hosting site Patch-Tag.

This release contains:

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

Further reading

http://patch-tag.com/repo/crypto

5.8.2 The Haskell ASN.1 Library

Report by:	Dominic Steinitz

We are still working on a complete restructuring of the library. Over 620 darcs patches have been committed to support the Packed Encoding Rules (PER) using a GADT to represent the Abstract Syntax Tree.

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

The currently release still supports BER for:

- X.509 identity certificates
- X.509 attribute certificates
- PKCS#8 private keys
- PKCS#1 version 1.5

Further reading

http://haskell.org/asn1.

5.8.3 MultiSetRewrite

Report by:	Martin Sulzmann

The MultiSetRewrite project is under active development (latest update March'09). MultiSetRewrite is a Haskell library extension to support multi-set rewrite rules with guards. The MultiSetRewrite library forms the core of

- a Constraint Handling Rules (CHR) solver,
- an extension of join patterns with guards, propagation and a parallel execution model (available as a separate package).

Further reading

- http://sulzmann.blogspot.com/2008/10/multi-set-rewrite-rules-with-guards-and.html
- http://hackage.haskell.org/cgi-bin/hackage-scripts/package/multisetrewrite

5.8.4 Graphalyze

Report by:	Ivan Lazar Miljenovic
Status:	version 0.5

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

Graphalyze is used in SourceGraph (→ 4.3.7), and was developed as part of my Mathematics Honours' thesis, *Graph Theoretic Analysis of Relationships Within Discrete Data*. The focus on this thesis was to develop computational tools to allow people to analyze discrete data sets. The output produced when running SourceGraph on Graphalyze can be found at http://community.haskell.org/~ivanm/Graphalyze/Graphalyze.html.

Further reading

- http://hackage.haskell.org/cgi-bin/hackage-scripts/package/Graphalyze
- http://code.haskell.org/Graphalyze
- http://ivanmiljenovic.files.wordpress.com/2008/11/honoursthesis.pdf

5.8.5 Takusen

Report by:	Alistair Bayley
Participants:	Oleg Kiselyov
Status:	active development

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

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

Currently we fully support Oracle, Sqlite, and PostgreSQL. ODBC support is nearly complete; string output bind-variables do not marshal correctly.

Things have been quiet. Since the last report we have fixed a few bugs and got the install process working with ghc-6.10.

Further reading

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

5.9 Generic and Type-Level Programming

5.9.1 uniplate

Report by:	Neil Mitchell

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

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

Further reading

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

5.9.2 Scrap Your Boilerplate (SYB)

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

Scrap Your Boilerplate (SYB) is a library for generic programming in Haskell. It has been supported by GHC since the 6.0 release. The library is based on

combinators and a few primitives for type-safe casting and processing constructor applications.

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

Recent changes

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

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

Future plans

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

Contact

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

Further reading

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

5.9.3 Extensible and Modular Generics for the Masses (EMGM)

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

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

Introduction

EMGM is a library for of datatype-generic programming using type classes. We represent Haskell datatypes as values using a sum-of-products structure representation. The foundation of EMGM allows programmers to write generic functions by induction on the structure of datatypes. The use of type classes in EMGM allows generic functions to support ad-hoc cases for arbitrary datatypes.

The library provides a sizable (and constantly growing) collection of ready-to-use generic functions. Here are some examples of these functions:

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

EMGM also comes with support for standard datatypes such as lists, `Either`, `Maybe`, and tuples. Adding support for your own datatype is straightforward using our new deriving API.

Background

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

Recent Development

Since the last publication of this report, EMGM has seen significant development. The most important is in

deriving representations for datatypes. Previously, library users would need to write their own structure representation and instances. Now, it is simply a matter of using a collection of Template Haskell functions. The deriving API provides both the single function `derive` for ease-of-use and a very flexible set of functions to allow programmers to choose exactly what they want.

The documentation of EMGM has seen major expansion and improvement. If you find something that could be better documented, we would like to know.

Other changes include some new functions. The function `bimap` allows one to map over bifunctors. The functions `everywhere` and `everywhere'` are similar to functions found in the Scrap Your Boilerplate library. They apply a transformation everywhere a certain type is found in a value.

Future plans

We are continuing to develop more generic functions and to explore the use of this library in many domains. There should be one or two releases before the next report. We welcome ideas or contributions from the community.

Contact

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

Further reading

More information can be found on the EMGM website. Download and browse the API of the library at the Hackage page for EMGM.

5.9.4 multirec: Generic programming with systems of recursive datatypes

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

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

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

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

Features and limitations

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

Future plans

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

Contact

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

Further reading

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

5.9.5 Generic rewriting library for regular datatypes

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

This library provides a set of functions to apply rewrite rules to a datatype. The rewrite functions are generic, so it is not necessary to re-implement the matching

and substitution machinery for every datatype. Additionally, the library provides a set of generic traversal functions that can be used together with rewriting.

Features and limitations

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

Future plans

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

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

Contact

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

Further reading

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

5.9.6 2LT: Two-Level Transformation

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

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

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

- A library of two-level transformation combinators. These combinators are used to compose transformation systems which, when applied to an input type, produce an output type together with the conversion functions that mediate between input and output types.
- Front-ends for VDM-SL, XML, and SQL. These front-ends support (i) reading a schema, (ii) applying a two-level transformation system to produce a new schema, (iii) converting a document/database corresponding to the input schema to a document/database corresponding to the output schema, and *vice versa*.
- A combinator library for transformation of point-free and structure-shy functions. These combinators are used to compose transformation systems for optimization of conversion functions, and for migration of queries through two-level transformations. Independently of two-level transformation, the combinators can be used to specialize structure-shy programs (such as XPath queries and strategic functions) to structure-sensitive point-free form, and *vice versa*.
- Support for schema constraints using point-free expressions. Constraints present in the initial schema are preserved during the transformation process and new constraints are added in specific transformations to ensure semantic preservation. Constraints can be simplified using the already existent library for transformation of point-free functions.

The various sets of transformation combinators are reminiscent of the combinators of Strafunski and the Scrap-your-Boilerplate (→ 5.9.2) approach to generic functional programming.

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

Future plans

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

Further reading

- Project URL: http://2lt.googlecode.com
- Alcino Cunha, José Nuno Oliveira, Joost Visser. *Type-safe Two-level Data Transformation*. Formal Methods 2006.
- Alcino Cunha, Joost Visser. Strongly Typed Rewriting For Coupled Software Transformation. RULE 2006.

- Pablo Berdaguer, Alcino Cunha, Hugo Pacheco, Joost Visser. *Coupled Schema Transformation and Data Conversion For XML and SQL*. PADL 2007.
- Alcino Cunha and Joost Visser. *Transformation of Structure-Shy Programs, Applied to XPath Queries and Strategic Functions*. PEPM 2007.
- Tiago L. Alves, Paulo Silva and Joost Visser. *Constraint-aware Schema Transformation*. RULE, 2008.

5.9.7 Data.Label — "atoms" for type-level programming

Report by:	Claus Reinke
Status:	experimental

A common problem for type-level programming (extensible record libraries, type-level numbers, ...) in Haskell is where to define shared atomic types (record field labels, type tags, type numerals):

- identical types defined in separate modules are not compatible
- common imports defining common types for several projects hurt modularity
- SML-style parameterized modules and type-sharing are not directly available

Using Template Haskell, and QuasiQuotes in particular, we can now at least work around this issue, by splitting the atoms:-) `Data.Label` provides type letters and combinators for constructing typed "atoms" from these letters, as well as quasiquoting and `Show` instances to hide some of this internal structure.

```
*Main> [$l|label|]
label
*Main> :t [$l|label|]
[$l|label|] :: Ll :< (La :< (Lb :< (Le :< Ll)))
```

This workaround lets users choose between shared or locally defined labels:

```
module A where
import Data.Label
data MyLabel
x = [$l|label|]
y = undefined::MyLabel
```

```
module B where
import Data.Label
data MyLabel
x = [$l|label|]
y = undefined::MyLabel
```

```
module C where
import Data.Label
import A
import B
ok = [A.x,B.x]
fails = [A.y,B.y]
```

It does so by offering a meta-level commonality: `A` and `B` do not have to agree on a common module to declare all their common types (`Data.Label` is unaffected by the specific labels its importers might use), they only need to agree on a common way of declaring all their sharable "atomic" types.

Further reading

- Example code: http://community.haskell.org/~claus/misc/labels.hs http://community.haskell.org/~claus/misc/Data/Label/TH.hs http://community.haskell.org/~claus/misc/Data/Label.hs
- Discussion: http://www.haskell.org/pipermail/haskell-cafe/2009-April/059819.html

5.10 User interfaces

5.10.1 Gtk2Hs

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

Gtk2Hs is a set of Haskell bindings to many of the libraries included in the Gtk+/Gnome platform. Gtk+ is an extensive and mature multi-platform toolkit for creating graphical user interfaces.

GUIs written using Gtk2Hs use themes to resemble the native look on Windows and, of course, various desktops on Linux, Solaris, FreeBSD, and Mac OS X using X11.

Gtk2Hs features:

- Automatic memory management (unlike some other C/C++ GUI libraries, Gtk+ provides proper support for garbage-collected languages)
- Unicode support
- High quality vector graphics using Cairo
- Cross-platform, multi-format multimedia playback with GStreamer
- Extensive reference documentation
- An implementation of the "Haskell School of Expression" graphics API
- Support for the Glade visual GUI builder
- Bindings to some Gnome extensions: GIO/GVfs, GConf, GtkSourceView 1.0 and 2.0, and the GtkMozEmbed widget for embedded web browsers
- An easy-to-use installer for Windows
- Packages for Fedora, Gentoo (→ 2.9.1), Debian, and FreeBSD
- New features:
  - Model-view widgets with stores being implemented in Haskell

– Full drag-and-drop and clipboard support

– Binding to GtkSourceView 2

– Support for GHC 6.10 and newer

– New demos and example code

The Gtk2HS library is continually being improved with new bindings, documentation, and bug fixes. Outside contributions are always welcome! We have recently released version 0.10.0, and are in the process of packaging 0.10.1 to be released soon.

In the future we hope to modularize Gtk2Hs and enable it to be built and distributed with Cabal and Hackage. This will enable people to just create, e.g., high-quality PDF documents using Cairo and Pango, performing image manipulation using Pixbuf and more. We also plan to bind more of the Gnome platform libraries, to allow compliant Gnome applications to be built entirely in Haskell.

Further reading

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

5.10.2 HQK

Report by:	Wolfgang Jeltsch
Participants:	Thomas Mönicke
Status:	provisional

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

We plan to develop a HQK-based UI backend for the Functional Reactive Programming library Grapefruit (→ 6.5.1).

Further reading

http://haskell.org/haskellwiki/HQK

5.10.3 wxHaskell

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

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

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

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

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

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

- Release of version 0.10.3, including binary installers. This added support for recent versions of the underlying wxWidgets library and for recent versions of GHC. Prior to this release, getting started with wxHaskell was becoming difficult for new users.
- Support for many new widget types. We now support almost all of the widgets provided in the wxWidgets distribution, including user-contributed widgets.
- Many bugfixes.
- Addition of support for XML descriptions of GUI layouts using XRC. This has been the most requested feature by users for some time, and has just been committed to the Darcs repository. We anticipate a new release with binary installers in the next few weeks, to make this feature more widely available.
- Work on porting to the GHC 6.10 release — wxHaskell in Darcs repository compiles out of the box on 6.10 release candidates.
- Improvements to Cabal support, with the intention of providing Cabal installable packages for wxcore and wx.

- Main repository moved from darcs.haskell.org to code.haskell.org.

- Dropped support for versions of wxWidgets prior to 2.8. We currently support building against wxWidgets 2.8.x and 2.9.x. This has resulted in a (small) number of backward incompatible API changes, compared to earlier versions.

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

Further reading

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

5.10.4 Shellac

Report by:	Robert Dockins
Status:	beta, maintained

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

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

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

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

Further reading

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

5.10.5 Haskeline

Report by:	Judah Jacobson
Status:	active development

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

In its latest release (0.6.1.3), Haskeline supports Unicode and runs both on the native Windows console and on POSIX-compatible systems. Its rich line-editing interface is user-customizable and includes emacs and vi modes, history recall and incremental search, undo support, and custom tab completion functions.

Recent work has extended compatibility to a wide variety of platforms and environments. Plans for further development include adding even more features to the user interface.

Further reading

- Home page: http://trac.haskell.org/haskeline
- Releases: http://hackage.haskell.org/cgi-bin/hackage-scripts/package/haskeline

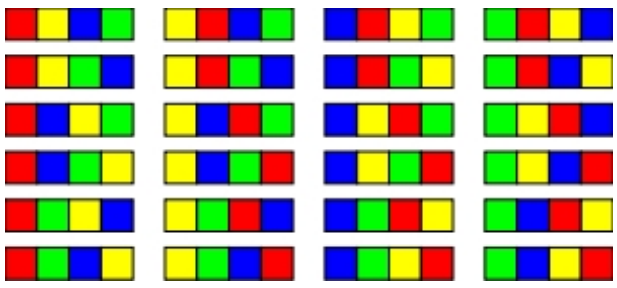
5.11 Graphics

5.11.1 diagrams

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

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

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



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

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

Further reading

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

5.11.2 FieldTrip

Report by:	Conal Elliott
Status:	active development

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

Surfaces are described as functions from 2D space to 3D space. As such, they are intrinsically curved rather than faceted. Surface rendering tessellates adaptively, caching tessellations in an efficient, infinite data structure (from the MemoTrie library (→ 5.7.3)) for reuse. Surface normals are computed automatically and exactly, using the derivative tools in the vector-space library (→ 5.6.3).

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

Further reading

http://haskell.org/haskellwiki/FieldTrip

5.11.3 LambdaCube

Report by:	Csaba Hruska
Status:	active development

LambdaCube aims to be the first general purpose 3D rendering engine written in a functional language. The main goal of this project is to provide a modern and feature rich graphical backend for various haskell projects (e.g., FRP libraries). The engine uses the same model and material format as Ogre3D (http://www.ogre3d.org). Currently it supports some basic features such as model loading, multitexturing, shaders, and resource management. The existing code uses OpenGL, and it runs at nearly same speed as a C/C++ rendering engine. The project is under heavy development.

Further reading

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

5.12 Music

5.12.1 Haskore revision

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

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

1. The `Music` data type has been generalized in the style of Hudak's "polymorphic temporal media." It has been made abstract by providing functions that operate on it.

2. The notion of instruments is now very general. There are simple predefined instances of the `Music` data type, where instruments are identified by Strings or General MIDI instruments, but any other custom type is possible, including types with instrument specific parameters.

3. Creation of CSound orchestra files in a functional style including feedback and signal processors with multiple outputs.

4. Support for the software synthesizer SuperCollider both in real-time and non-real-time mode through the Haskell interface by Rohan Drape.

5. Conversion between MIDI file and Haskore representation of Music. Real-time MIDI is supported via ALSA on Linux.

6. A package for lists of events with time information has been factored out, as well as a package for non-negative numbers, which occur as time differences in event lists.

7. Support for infinite `Music` objects is improved. CSound may be fed with infinite music data through a pipe, and an audio file player like Sox can be fed

with an audio stream entirely rendered in Haskell. (See Audio Signal Processing project (→ 6.6.1).)

Future plans

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

Further reading

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

5.12.2 Euterpea

Report by:	Paul Hudak
Participants:	Eric Cheng, Paul Liu, Donya Quick
Status:	experimental, active development

Euterpea is a new Haskell library for computer music applications. It is a descendent of Haskore and HasSound, and is intended for both educational purposes as well as serious computer music development. Euterpea is a "wide-spectrum" library, suitable for high-level music representation, algorithmic composition, and analysis; mid-level concepts such as MIDI; and low-level audio processing, sound synthesis, and instrument design. It also includes a "musical user interface", a set of computer-music specific GUI widgets such as keyboards, guitar frets, knobs, sliders, and so on. The performance of Euterpea is intended to be as good or better than any existing computer music language – it can be used for real-time applications, not just using MIDI, but also using a high-performance back-end for real-time audio.

Euterpea is being developed at Yale in Paul Hudak's research group, where it has become a key component of Yale's new Computing and the Arts major. Hudak is teaching a two-term sequence in computer music using Euterpea, and is developing considerable pedagogical material, including a new textbook tentatively titled "The Haskell School of Music". The name "Euterpea" is derived from "Euterpe", who was one of the nine Greek Muses (goddesses of the arts), specifically the Muse of Music.

History

Haskore is a Haskell library developed almost 15 years ago by Paul Hudak at Yale for high-level computer music applications. HasSound is a more recent Haskell library developed at Yale that serves as a functional front-end to csound's sound synthesis capabilities. Haskore and HasSound have evolved in a number of different ways over the years, most notably through Henning Thielemann's darcs library for Haskore, to which many people have contributed. There are many good ideas in that library, but it has become overly complex and lacks a coherent design concept.

Future plans

The Euterpea developers' plan is to shamelessly steal good ideas from these previous efforts, integrate them into a coherent new framework, remove dependencies from as many non-Haskell libraries as possible, add new features such as musical GUI widgets, and incorporate new methods for high-performance stream processing recently developed at Yale, to make Euterpea the library of choice for discriminating computer music hackers.

Further reading

- http://plucky.cs.yale.edu/cs431
- http://plucky.cs.yale.edu/cs431/HaskoreSoeV-0.12.pdf

5.13 Web and XML programming

5.13.1 Haskell XML Toolbox

Report by:	Uwe Schmidt
Status:	seventh major release (current release: 8.3.0)

Description

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

The Haskell XML Toolbox is based on the ideas of HaXml (→ 5.13.2) and HXML, but introduces a more general approach for processing XML with Haskell. The processing model is based on arrows. The arrow interface is more flexible than the filter approach taken in the earlier HXT versions and in HaXml. It is also safer; type checking of combinators becomes possible with the arrow approach.

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

Features

- Validating XML parser
- Very liberal HTML parser
- Lightweight lazy parser for XML/HTML based on Tagsoup (→ 5.13.3)

- Easy de-/serialization between native Haskell data and XML by pickler and pickler combinators
- XPath support
- Full Unicode support
- Support for XML namespaces
- Cabal package support for GHC
- HTTP access via Haskell bindings to libcurl
- Tested with W3C XML validation suite
- Example programs
- Relax NG schema validator
- An HXT Cookbook for using the toolbox and the arrow interface
- Basic XSLT support
- Darcs repository with current development version (8.3.1) under http://darcs2.fh-wedel.de/hxt

Current Work

Currently mainly maintenance work is done. This includes space and runtime optimizations, the internal representation of XML names has been changed to gain less memory consumption. Equal XML names share the same main memory.

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

The HXT library is extensively used in the Holumbus project (→ 6.3.1), there it forms the basis for the index generation.

Further reading

The Haskell XML Toolbox Web page (http://www.fh-wedel.de/~si/HXmlToolbox/index.html) includes downloads, online API documentation, a cookbook with nontrivial examples of XML processing using arrows and RDF documents, and master theses describing the design of the toolbox, the DTD validator, the arrow based Relax NG validator, and the XSLT system.

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

5.13.2 HaXml

Report by:	Malcolm Wallace
Status:	stable, maintained

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

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

The lazy parser combinators used by HaXml now live in a separate library package called polyparse.

Further reading

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

5.13.3 tagsoup

Report by:	Neil Mitchell

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

The library provides a basic data type for a list of unstructured tags, a parser to convert HTML into this tag type, and useful functions and combinators for finding and extracting information. The library has seen real use in an application to give Hackage (→ 5.1) listings, and is used in the next version of Hoogle (→ 4.4.1).

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

Further reading

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

5.14 System

5.14.1 hinotify

Report by:	Lennart Kolmodin
Status:	alive

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

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

Further reading

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

5.14.2 hlibev

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

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

Further reading

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

6 Applications and Projects

6.1 For the Masses

6.1.1 Darcs

Report by:	Eric Kow
Participants:	Ganesh Sittampalam, Trent Buck, Gwern Branwen
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 release, darcs 2.2, was in January 2009. This release provides improved support for Windows, a few low-level optimizations and a preliminary version of the darcs library. It is also the first of our biannual releases, with the next release scheduled for July.

Meanwhile, we have been continuing to build up the darcs community and are encouraged the developments from the past 6 months:

1. Fundraising: In a recent fundraising drive, we managed to raise $1072 (USD) to help pay for travel to the our second darcs hacking sprint, held as part of the Haskell Hackathon. Using these funds, we subsidized travel costs to the Utrecht sprint for three of the participating Darcs hackers, and we have some left over the next sprint.

2. Software Freedom Conservancy: The Software Freedom Conservancy (SFC) is an organization composed of Free and Open Source Software projects. Joining the SFC allows darcs to receive tax-deductible donations in the United States and to hold assets, and also provides some protection to our developers from personal liability.

3. Google Summer of Code: Darcs developer Petr Rokai has been selected to participate in the 2009 Google Summer of Code program. His project will optimize darcs' use of hashed storage repositories, making Darcs faster and more scalable. We are extremely grateful for generous support of the Haskell.org mentoring organization, which has provided us with one of their Google Summer of Code slots.

4. Hosting: Last but not least, Thomas Hartman and Matthew Elder have begun a commercial darcs hosting venture called Patch-tag. Patch-tag offers free darcs hosting to open source projects, and paid hosting for others. We wish them luck!

We are excited about these recent developments and their potential to help us continue to steadily improve darcs. 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.

Further reading

http://darcs.net

6.1.2 xmonad

Report by:	Don Stewart
Status:	active development

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

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

Further reading

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

6.2 Education

6.2.1 Exercise Assistants

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

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

For each exercise domain, we need the same functionality, such as unifying and rewriting terms, generating exercises, traversing terms, and testing for (top-level) equality of two terms. For these parts we are currently using the generic programming libraries Uniplate and Multirec, which help us to reduce code size and improve the reliability of our code. We have reported our experiences with generic programming for domain reasoners, and identified some missing features in the libraries. Fully exploiting generic programming techniques is ongoing work.

We have recently integrated our tools with the Digital Mathematics Environment (DWO) of the Freudenthal Institute. This environment contains a collection of applets for practicing exercises in mathematics. A selected number of applets has been extended with our facility to automatically generate hints and worked-out examples, and the first results are promising. To offer this service, we have introduced views for mathematical expressions, and combined these with our rewriting technology. A view specifies a canonical form, and abstracts over a set of algebraic laws. Our views are based on the views proposed by Wadler.

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

Further reading

- http://ideas.cs.uu.nl/trac
- Bastiaan Heeren and Johan Jeuring. Canonical Forms in Interactive Exercise Assistants. To appear in Mathematical Knowledge Management (MKM 2009).
- Johan Jeuring, José Pedro Magalhães, and Bastiaan Heeren. Experience Report: Generic Programming for Domain Reasoners. Draft version.
- Alex Gerdes, Bastiaan Heeren, and Johan Jeuring. Constructing Strategies for Programming. Proceedings of the International Conference on Computer Supported Education (CSEDU 2009).

6.2.2 Holmes, plagiarism detection for Haskell

Report by:	Jurriaan Hage
Participants:	Brian Vermeer

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

We are currently at the stage that metrics can be implemented on top of the Helium front-end. Many of these metrics will be defined on an auxiliary structure, the function-call flow graph. Dead-code removal has taken place, fully qualified names are used throughout, and template removal is now easily possible.

6.2.3 Lambda Shell

Report by:	Robert Dockins
Status:	beta, maintained

The Lambda Shell is a feature-rich shell environment and command-line tool for evaluating terms of the pure, untyped lambda calculus. The Lambda Shell builds on the shell creation framework Shellac (→ 5.10.4), and showcases most of Shellac's features.

Features of the Lambda Shell include:

- Evaluate lambda terms directly from the shell prompt using normal or applicative order. In nor-

mal order, one can evaluate to normal form, head normal form, or weak head normal form.

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

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

Further reading

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

6.2.4 INblobs — Interaction Nets interpreter

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

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

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

Features

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

Further reading

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

6.2.5 Soccer-Fun

Report by:	Peter Achten
Status:	active development

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

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

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

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

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

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

Future plans

There are many plans for the future:

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

Further reading

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

6.3 Web Development

6.3.1 Holumbus Search Engine Framework

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

Description

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

An instance of the Holumbus framework is the Haskell API search engine Hayoo! (http://holumbus.fh-wedel.de/hayoo/). The web interface for Hayoo! is implemented with the Janus web server, written in Haskell and based on HXT (→ 5.13.1).

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
- Darcs repository with current development version under http://darcs2.fh-wedel.de/holumbus
- Distributed building of search indexes

Current Work

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

Stefan Schmidt has finished his master thesis developing the Holumbus MapReduce system, a framework

for distributed computing with an architecture like the Google *map–reduce* system.

A follow-up thesis bringing this MapReduce system into a development status for real world distributed applications has been started by Sebastian Reese. One subgoal of this work is to write a *cookbook* for programming with the MapReduce framework and for giving tuning and configuration hints. The distributed recomputation and update of the Hayoo! index will be one of the real world test cases of this project.

Further reading

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

6.3.2 Top Writer

Report by:	Jon Strait
Status:	experimental, active development

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

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

Future plans

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

Further reading

http://www.moonloop.net/topwriter

6.3.3 Bamboo blog engine (previously: Panda) / Hack Webserver interface

Report by:	Jinjing Wang
Status:	experimental

Bamboo is an upgrade to the Panda blog engine. While keeping the same features from Panda, it runs on multiple server backend, including Kibro / fcgi and Happstack-server.

Hack Webserver interface plays a crucial rule in realizing the portability of Bamboo. Hack is a port of the Ruby's Rack web server interface, which is inspired by Python's WSGI. Hack's specification is defined in a simple haskell data structure, that can be plugged into different server handlers. Changing a server backend is as simple as changing the name of the server handler module which has been imported.

Bamboo is free software under GPL, future development will focus on extending features by means of Hack middleware.

Further reading

- http://rack.rubyforge.org/
- http://github.com/nfjinjing/bamboo
- http://github.com/nfjinjing/hack
- http://chrisdone.com/blog/tags/Kibro.html
- http://happstack.com/

6.3.4 InputYourData.com

Report by:	Enzo Haussecker
Status:	beta

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

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

I am now interested in developing a similar type of website, except Wiki style — where all web applications are created by the user. Essentially, I am designing a web application where users can symbolically

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

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

6.3.5 Hircules

Report by:	Jens Petersen
Status:	hibernating

Hircules is an IRC client built on top of gtk2hs (→ 5.10.1).

No new changes released since last time. I am in the progress of adding cabal and hierarchical modules.

I am still hoping to import the code to code.haskell.org soon and finally make a new release for ghc-6.10 on Hackage. I still would like to add support for multiple irc connections and get other people involved.

Further reading

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

6.3.6 HCluster

Report by:	Alberto Gómez Corona

HCluster (provisional name) aims to be a massive remote clustering middleware that permits:

- distributed transactions between connected nodes in the Internet
- work with nodes online as well as offline + synchronization
- hot plug-in of nodes
- no master nodes, no single point of failure/control
- theoretical massive scalability, reliability, availability

HCLuster idea was born as a solution for the problem of verifiability in electronic democracy: How to avoid tampering of the servers and/or results? By means of a loosely coupled network of remote servers that execute the same software, can be connected and disconnected at any time, handle the same data and produce the same result. Thus, it must work in the Internet, can handle connection errors, re-synchronization and support a great number of nodes. Splits and rejoin of branches must be supported too. Essentially, it must support that anyone interested can plug its personal computer and repeat the process being verified.

This solution for plain verifiability also provides availability in ordinary applications. Any node can initiate a process (that may involve a transaction, a query, a calculation etc.). The design of synchronization permits nodes to work in online as well as offline mode + periodic synchronization with certain restrictions. This is achieved by making visible, to the application developer, the distinction between synchronous and asynchronous transactions.

Theoretically, the synchronization algorithm accepts distributed transactions, so that, for example, some nodes can have data1 and others data2 and still process transactions that handle data1 and data2 simultaneously. The distribution is transparent to the programmer, so re-locations of data can be done among the nodes.

Finished basic services: HTTP protocol, reconnection, synchronization.

Future plans

To finish the design of higher level services: distributed transactions. To test the synchronization services. To create internet documentation.

Contact

⟨agocorona@gmail.com⟩

6.3.7 JavaScript Monadic Writer

Report by:	Dmitry Golubovsky
Status:	active development

JavaScript Monadic Writer (JSMW) is an extensible monadic framework on top of the Haskell DOM bindings. It provides an EDSL to encode JavaScript statements and expressions in typesafe manner. It borrows some ideas from HJScript, but uses slightly different EDSL notation.

The idea behind JSWM is to provide an intermediate form that could be used as an end-point for conversion from Haskell Core. The EDSL however may be considered as a programming tool on its own. While the EDSL alone is not sufficient for translation of an

arbitrary Haskell program to JavaScript, Haskell type system is still available to help produce correct code.

Further reading

The jsmw package on Hackage
http://hackage.haskell.org/cgi-bin/hackage-scripts/package/jsmw

6.3.8 Haskell DOM Bindings

Report by:	Dmitry Golubovsky
Status:	active development

Haskell DOM bindings is a set of monadic smart constructors on top of the WebBits representation of JavaScript syntax to generate JavaScript code that calls DOM methods and accesses DOM objects' attributes.

In order to represent the hierarchy of DOM interfaces, Haskell type classes are used. For example, for the interfaces `Node` and `Document` (the latter inherits from the former) there are two classes: `CNode` and `CDocument`. Also, for each DOM interface, a phantom data type is defined: `TNode`, and `TDocument` in this case. Phantom types represent concrete values (references to DOM objects) while type classes are used for type constraints in functions working with DOM objects. The CDocument class is defined as:

```
class CNode a => CDocument a
data TNode
data TDocument
instance CNode TNode
instance CDocument TDocument
instance CNode TDocument
```

Type constraints are used to define methods of each class, e. g.

```
hasChildNodes :: (Monad mn, CNode this)
  => Expression this -> mn (Expression Bool)
```

so, `hasChildNodes` can be called on both `Node` and `Document`, but

```
createElement :: (Monad mn, CDocument this,
                  CElement zz)
  => Expression String -> Expression this
     -> mn (Expression zz)
```

only on nodes representing documents.

The bindings were auto-generated from OMG IDL files provided by the Web Consortium. The IDL to Haskell converter is based on H/Direct IDL parser. Automatic IDL conversion is expected to simplify Haskell Web development because of the large number of methods and attributes defined in contemporary DOM whose type signatures are hard and time-consuming to derive manually.

Further reading

- Document Object Model (DOM) Technical Reports
  http://www.w3.org/DOM/DOMTR
- The DOM package on Hackage
  http://hackage.haskell.org/cgi-bin/hackage-scripts/package/DOM

6.4 Data Management and Visualization

6.4.1 Pandoc

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

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

Since the last report, there have been two releases of pandoc (1.1 and 1.2), including many bug fixes and the following new features:

- Support for literate Haskell.
- New `-jsmath` and `-email-obfuscation` options.
- Better CSS styling in HTML tables.
- Windows installer no longer requires admin privileges.
- Support for citeproc-hs-0.2.

Further reading

http://johnmacfarlane.net/pandoc/

6.4.2 tiddlyisar

Report by:	Slawomir Kolodynski
Status:	under development

tiddlyisar is a tool for generating TiddlyWiki renderings of IsarMathLib source. IsarMathLib is a library of mathematical proofs formally verified by the Isabelle/ZF theorem proving environment. The tiddlyisar tool parses IsarMathLib source and generates TiddlyWiki markup text. The generated view features jsMath based mathematical symbols, cross referenced theorems, and structured proofs expanded on request.

The rendering can be viewed on the Tiddly Formal Math site. tiddlyisar is included in the IsarMathLib distribution under GPLv3 license. The source can be browsed at the IsarMathLib Subversion repository URL provided below.

Further reading

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

6.4.3 HaExcel — From Spreadsheets to Relational Databases and Back

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

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

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

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

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

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

Further reading

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

6.4.4 Between Types and Tables

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

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

Future plans

Currently, the library is a proof of concept that only works with Clean on Linux and a MySQL database. However, we intend to integrate this library with the work on dynamic workflow specifications in the iTask system (→ 6.8.3) somewhere in the (near) future.

Further reading

- http://www.st.cs.ru.nl/papers/2008/lijb08-BetweenTypesAndTablesMasterThesis.pdf
- http://www.st.cs.ru.nl/papers/2008/lijb08-BetweenTypesAndTables-IFL2008-DRAFT.pdf

6.4.5 SdfMetz

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

SdfMetz supports grammar engineering by calculating grammar metrics and other analyses. Currently it supports four different grammar formalisms (SDF, DMS, Antlr, and Bison). The category of supported metrics are size, complexity, structural and disambiguation. The disambiguation metrics are applicable to the SDF formalism only. Metrics output is a textual report or in Comma Separated Value format. The additional analyses implemented are graph visualization of the immediate successor graph, transitive closure graph and strongly-connected components graph outputted in DOT format, and visualization of the non-singleton levels of a grammar.

The definition of all except the ambiguity and the NPath metrics were taken from the paper *A metrics suite for grammar based-software* by James F. Power and Brian A. Malloy. The ambiguity metrics were defined by the tool author exploiting specific aspects of SDF grammars, and the NPath metric definition was

taken from the paper *NPATH: a measure of execution path complexity and its applications*.

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

Future plans

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

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

Further reading

The web site of SdfMetz (http://sdfmetz.googlecode.com) includes the tool source code, and pointers to relevant work about grammar metrication.

6.4.6 The Proxima 2.0 generic editor

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

Proxima 2.0 is an open-source web-based version of the Proxima generic presentation-oriented editor for structured documents.

- Proxima is a *generic* editor. This means that the editor can be instantiated for arbitrary document types, supplemented by parser and presentation sheets. The content of a Proxima document can be mixed text, images and diagrams.
- Proxima is a *presentation-oriented editor*. This means that the user performs the edit operations on the WYSIWYG screen presentation of the document.
- Proxima is aware of the structure of the document. Even while editing the presentation of the document, the edit operations can be structural. For example, a section can be changed into a subsection.

Another feature of Proxima is that it offers generic support for specifying content-dependent computations. For example, it is possible to create a table of contents of a document that is automatically updated as chapters or sections are added or modified.

Proxima 2.0

Proxima 2.0 provides a web-interface for Proxima. Instead of rendering the edited document onto an application window, Proxima 2.0 is a web-server that sends an HTML rendering of the document to a client. The client catches mouse and keyboard events, and sends these back to the server, after which the server sends incremental rendering updates back to the client. As a result, advanced editors can be created, which run in any browser.

Future plans

Proxima 2.0 is an open source project. A basic prototype has been constructed, which yields promising results. Still, a lot of work needs to be done on the underlying Proxima system, as well as on methods for handling latency and low bandwidth. We are looking for people who would like to participate in the project.

Further reading

http://www.oblomov.biz/proxima2.0.html

6.5 Functional Reactive Programming

6.5.1 Grapefruit

Report by:	Wolfgang Jeltsch
Status:	provisional

Grapefruit is a library for Functional Reactive Programming (FRP) with a focus on user interfaces. FRP makes it possible to implement reactive and interactive systems in a declarative style. With Grapefruit, user interfaces are described as networks of communicating widgets and windows. Communication is done via different kinds of signals which describe temporal behavior.

There was a first Grapefruit release in February 2009. Since then, Grapefruit was improved notably. This article describes the current development version.

Grapefruit consists of several packages, each having interesting features:

grapefruit-frp core support for FRP

- proper handling of simultaneous events
- incrementally updating signals of sequences and sets
- out-of-the-box signal memoization
- protection against signal values depending on starting time
- scalable implementation

grapefruit-records a record system

- optional fields
- record reduction and field reordering through pattern matching

- general record types which are specialized through "styles"

grapefruit-ui general support for UI programming

- support for different backends for using different UI toolkits through the same API
- backend selection at compile time or runtime

grapefruit-ui-gtk UI backend based on Gtk2Hs (→ 5.10.1)

grapefruit-examples several little demos

We hope to provide the following features in the future:

- support for user interfaces with changing structure
- model-view-controller support for most widgets
- support for animated graphics
- UI backend based on HQK (→ 5.10.2)

Further reading

http://haskell.org/haskellwiki/Grapefruit

6.5.2 Reactive

Report by:	Conal Elliott
Status:	active development

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

- Much of the original interface is replaced by instances of standard type classes. In most cases, the denotational semantics of these instances is simple and inevitable, following from the principle of type class morphisms.
- The original idea of reactive behaviors is composed out of two simple notions:
  - *Reactive values* are temporally discrete and reactive. They have a purely data representation, and hence cache for free.
  - *Time functions* are temporally continuous and non-reactive.
- Reactive provides and builds on *functional futures*, which are time/value pairs with several handy type class instances. Futures allow one to conveniently compute with values before they can be known, with a simple, purely functional semantics (no IO). Futures are polymorphic over both values *and* time, requiring only that time is ordered.
- A particularly useful type of time, based on Warren Burton's "improving values", reveals partial information in the form of lower bounds and minima, before the times can be known precisely. (Semantically non-flat.)
- Improving values are implemented on top of a semantically simple "unambiguous choice" operator, (see unamb (→ 5.3.4)).
- Reactive manages (I hope) to get the efficiency of data-driven computation with a (sort-of) demand-driven architecture. For that reason, Reactive is garbage-collector-friendly.

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

Further reading

http://haskell.org/haskellwiki/Reactive

6.5.3 Functional Hybrid Modeling

Report by:	George Giorgidze
Status:	experimental

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

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

FHM is an approach to modeling which combines functional programming and non-causal modeling with the aim to improve state of the art of non-causal modeling languages. In particular, the FHM approach proposes modeling with first class model fragments (defined by continuous DAEs) using combinators for their composition and discrete switching. The discrete switching combinators enable modelling of hybrid systems (i.e. systems that exhibit both continuous and discrete dynamic behavior). The key concepts of FHM

originate from work on Functional Reactive Programming (FRP).

We are implementing Hydra, an FHM language, as a domain-specific language embedded in Haskell. The method of embedding employs quasiquoting, a new feature introduced in GHC 6.10, and enables modelers to use the domain specific syntax in their models. The present prototype implementation of Hydra enables modeling with first class model fragments, but only supports continuous systems. Currently, we are implementing discrete switching combinators and extending the simulator to enable modeling and simulation of highly dynamic hybrid systems.

Further reading

http://www.cs.nott.ac.uk/~ggg/

6.5.4 Elerea

Report by:	Patai Gergely
Status:	experimental

Elerea (Eventless reactivity) is a tiny continuous-time FRP implementation without the notion of event-based switching and sampling, with first-class signals (time-varying values). Reactivity is provided through a latching mechanism where a signal changes its behavior as dictated by a boolean input signal.

Elerea provides an easy to use applicative interface, supports recursive signals (a definition like `sine = integral 0 (integral 1 (-sine))` works without a hitch) and arbitrary external input. Cyclic dependencies are detected on the fly and resolved by inserting delays dynamically, unless the user does it explicitly.

The library is minimal by design, and it provides low-level primitives one can build a cleaner set of combinators upon. Also, it is relatively easy to adapt it to any imperative framework, although it is probably not a good choice to program primarily event-driven systems.

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.

Further reading

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

6.6 Audio and Graphics

6.6.1 Audio signal processing

Report by:	Henning Thielemann
Status:	experimental, active development

In this project, audio signals are processed using pure Haskell code and the Numeric Prelude framework (→ 5.6.2). The highlights are:

- a basic signal synthesis backend for Haskore (→ 5.12.1),
- 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 tried hard to preserve the functional style of programming and do not need Arrows and according notation.
- We checked several low-level implementations in order to achieve reasonable speed. We complement the standard list structure with a lazy `StorableVector` structure and a `StateT s Maybe a` generator, like in stream-fusion.
- 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.

Recent advances are:

- Novel algorithm for independently modulating time and phase of a sampled monophonic sound,
- Stand-alone binding to Sox for audio format conversion and playback,
- Package for fusion of stream-fusion list operations with storable vector that allows easy combination of the flexibility of the first one with the efficiency of the latter one,
- improved safety of causal processes with physical dimensions,
- enhanced type class framework for unifying signals expressed as lists, storable vectors and signal generators.

Further reading

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

6.6.2 hsProcMusic

Report by:	Stephen Lavelle
Status:	work ongoing

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

- Generate a 2-part melodic canon from a motivic seed. It implements a toy-version of first species counterpoint to generate canonical material.
- Analyze sets of melodies and produce candidates submelodies (including, optionally, slightly altered submelodies) for prospective polyphonic combination. This is related to a currently unrealized program to investigate cohomological aspects of counterpoint.
- Consonance-preserving map generator. Given two collections of pitch-classes, this program (with some harmonic assumptions) can generate all transformations from one to another that preserve relative consonance.
- A toy chord-progression generator based around Lerdahl's *Generative Theory of Tonal Music*. This is currently in active development, and I am aiming to bulk out its harmonic capabilities over the following months.

The source code can be downloaded from my website.

Further reading

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

6.6.3 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 (→ 5.3.2), etc.). Once appropriate geometric primitives have been extracted by the image processing wrappers we can define interesting computations using elegant functional constructions.

Recent developments include support for off-line processing, improved camera combinators, and a draft of the tutorial.

Further reading

http://www.easyVision.googlepages.com

6.6.4 photoname

Report by:	Dino Morelli
Status:	stable, maintained

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

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

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

Further reading

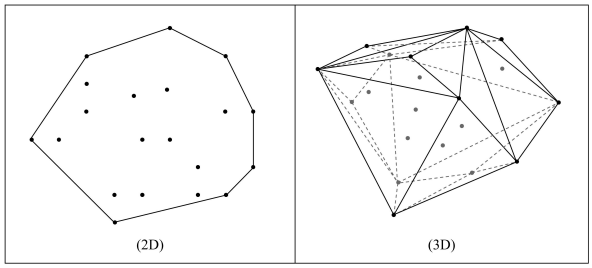
- Project page:
  http://ui3.info/d/proj/photoname.html
- Source repository:
  `darcs get` http://ui3.info/darcs/photoname

6.6.5 Simplex-Based Spatial Operations

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

The project is to implement spatial operations independent of dimension. There is much need in computational geometry and related fields to extend 2D spatial operations to 3D and higher dimensions. Approaches designed for a specific dimension lead to different implementations for different dimensions. Following such approaches, the code for a package that supports spatial operations for both 2D and 3D cases is nearly two times the code size for 2D. An alternative is dimension independent approaches. However, they are still implemented separately for each dimension. The main reason is lack of efficient data structures in the current programming languages. This research goes one step up the ladder of dimension independence in spatial operations. It implements dimension independent spatial operations using the concept of n-simplex. This n-dimensional data type is defined as a list, and its fundamental operations (e.g., dimension, orientation, boundary, clockwise and anticlockwise tests, etc.) are

developed as higher order functions over lists, which are treated efficiently in functional programming languages. Some spatial operations (e.g., distance and convex hull computations) have been implemented as case studies. A graphical user interface written with wxHaskell functions has been developed to illustrate the graphical results. The following figure shows the results of the convex hull computation for some 2D and 3D points.



Further reading

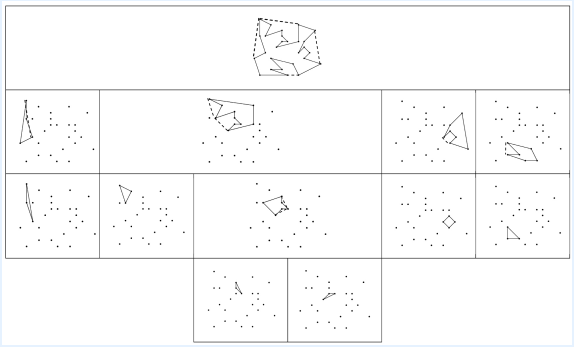
- F. Karimipour, M.R. Delavar, and A.U. Frank. A Mathematical Tool to Extend 2D Spatial Operations to Higher Dimensions, Proceedings of the International Conference on Computational Science and Its Applications (ICCSA 2008), (O. Gervasi, B. Murgante, A. Lagan, D. Taniar, Y. Mun, and M. Gavrilova, eds.),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

6.6.6 n-Dimensional Convex Decomposition of Polytops

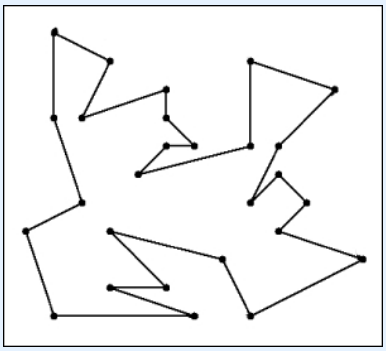
Report by:	Farid Karimipour
Participants:	Rizwan Bulbul, Andrew U. Frank
Status:	active development

This is the continuation of the work on "Simplex-based Spatial Operations" (→ 6.6.5), where we showed how to implement dimension independent spatial operations using the concept of n-simplexes. The results for n-dimensional convex hull computation were demonstrated through a graphical user interface written with wxHaskell functions. In this report, we have applied the same approach to a more complicated spatial analysis, i.e., convex decomposition of polytops. Convexity is a simple but useful concept. Convex objects are much easier to deal with in terms of storage and operations: the intersection of two convex objects is a convex object, the calculation of areas/volumes is straightforward and so is the "point-in-polygon" problem, etc.

There are several approaches to decompose a polytop to convex components, some of which may be adapted for different dimensions. However, they still require separate implementations for each dimension. The main reason is lack of suitable data structures in the current programming languages. We use the n-simplexes to implement an n-dimensional algorithm for convex decomposition of polytops. It builds a tree of signed convex components: components in even levels are additive, whereas components in odd levels are subtractive. An example figure:



The algorithm starts with placing the convex hull of the input polytop as the root of the tree. Subtraction of the computed convex hull from the input polytop yields a set of split polytops, which are the elements of the next level of the tree. The procedure applies to the non-convex elements and it repeats until all of the elements are convex. The list data structures and list operations are used for implementation of the algorithm. Polytops are represented as a list of n-simplexes, which are described as a list of points, per se. It allows us to describe the operations for convex decomposition of polytops as a combination of operations of n-simplexes, which turns to be operations on lists. Since the representation and operations are defined independent of dimension, the decomposition algorithm can be used for polytops of any dimension. The following figure shows a 2D example polytop which is decomposed to convex components as the above figure.



Further reading

see other entry (→ 6.6.5)

6.6.7 DVD2473

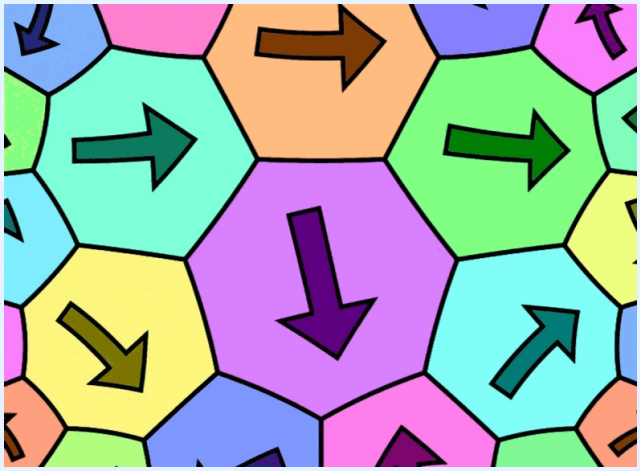
Report by:	Claude Heiland-Allen
Status:	complete

DVD2473 is a generative DVD video artwork.

The somewhat abstract title is a semi-literal description with all the descriptive elements taken out - it is a DVD of a perfect coloring in 24 colors of a 7,3 hyperbolic tiling. It uses the DVD virtual machine to navigate around the space at pseudo-random.

The mathematics (permutations and Moebius transforms) of the space were implemented in Haskell, with rendering with Gtk2Hs/Cairo including a little C code to copy and convert pixels from Cairo to ByteString. The video encoding used HSH to pipe ByteString to external tools. The XML control file for DVD authoring was generated with a separate Haskell program. A bash script automates building an iso from the source code directory.

There are no plans for further development, the artwork is finished.



Further reading

http://claudiusmaximus.goto10.org/cm/2009-01-07_dvd2473.html

6.7 Proof Assistants and Reasoning

6.7.1 Galculator

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

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

The prototype of *Galculator* is being developed under an ongoing PhD project. It is still experimental and things tend to change quickly. The current internal allows us to represent polymorphic proof-objects and having a type inference system to automatically infer their type representations. The details of implementation can be found in an article published in *PPDP'08*.

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

Currently, we are working on the automatic derivation of the so-called "free-theorems" of polymorphic functions (→ 3.3.1) and their application to proofs. Moreover, more complex constructions of Galois connections are also being studied. Finally, we plan to integrate the *Galculator* with a theorem prover, namely *Coq*.

Further reading

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

6.7.2 funsat: DPLL-style Satisfiability Solver

Report by:	Denis Bueno
Status:	version 0.6.0

Funsat is a native Haskell SAT solver that uses modern techniques for solving SAT instances. Our goal is to facilitate convenient embedding of a reasonably fast SAT solver as a constraint solving backend in other applications. Currently funsat can solve constraints expressed in conjunctive normal form (CNF) and as propositional logic circuits (new in 0.6) via an efficient conversion to CNF.

Funsat produces a total variable assignment for satisfiable inputs, and a resolution proof of unsatisfiability for unsatisfiable inputs. If unsatisfiable, funsat can also generate a small, unsatisfiable subformula of the input (unsatisfiable core). This latter feature is especially useful for reporting to a user why a complicated logical constraint was unsatisfiable.

We have tested funsat heavily. For each satisfiable formula, its assignment is checked; for each unsatisfiable formula, the resolution proof is checked. Currently funsat passes over 20,000 automated and over 3,000 static test cases.

Current implementation details include two-watched literals, conflict-directed learning, non-chronological backtracking, a VSIDS-like dynamic variable ordering, and restarts. Funsat can solve many structured instances from satlib (http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html) including PARITY (16 series), BF, blocksworld, and logistics. Many are solved in a few seconds.

Further reading

- http://hackage.haskell.org/cgi-bin/hackage-scripts/package/funsat
- The source code repository contains the latest funsat as well as many problem instances for benchmarking. It is available as a git repository, accessible like so: `$ git clone` git://github.com/dbueno/funsat.git

6.7.3 Saoithín: a 2nd-order proof assistant

Report by:	Andrew Butterfield
Status:	ongoing

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

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

A version of the software has been trialled out on 3rd-year students taking a Formal Methods elective course (https://www.cs.tcd.ie/Andrew.Butterfield/Teaching/3BA31/#Software) A first public release of the software under GPL will now happen during Summer 2009 — For now, Windows executables can be downloaded from the above link.

Further reading

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

6.7.4 Inference Services for Hybrid Logics

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

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

The Talaris group at Loria, Nancy, France (http://talaris.loria.fr) and the GLyC group at the Computer Science Department of the University of Buenos Aires, Argentina (http://www.glyc.dc.uba.ar/) are developing a suite of tools for automated reasoning for hybrid logics, available at http://code.google.com/p/intohylo/. Most of them are (successfully) written in Haskell. See HyLoRes (→ 6.7.5), HTab (→ 6.7.6), and HGen (→ 6.7.7).

6.7.5 HyLoRes

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

HyLoRes is an automated theorem prover for hybrid logics (→ 6.7.4) based on a resolution calculus. It is sound and complete for a very expressive (but undecidable) hybrid logic, and it implements termination strategies for certain important decidable fragments. The project started in 2002, and has been evolving since then. It is currently being extended to handle even more expressive logics (including, in particular, temporal logics). We have very recently added support for model-generation for satisfiable formulas.

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

Further reading

- Areces, C. and Gorin, D. *Ordered Resolution with Selection for H(@).* In Proceedings of LPAR 2004, pp. 125–141, Springer, Montevideo, Uruguay, 2005.
- Areces, C. and Heguiabehere, J. *HyLoRes: A Hybrid Logic Prover Based on Direct Resolution.* In Proceedings of Advances in Modal Logic 2002, Toulouse, France, 2002.
- Site and source: http://code.google.com/p/intohylo/

6.7.6 HTab

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

HTab is an automated theorem prover for hybrid logics (→ 6.7.4) based on a tableau calculus. It implements a terminating tableau algorithm for the basic hybrid logic extended with the universal modality.

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

Further reading

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

6.7.7 HGen

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

HGen is a random CNF (conjunctive normal form) generator of formulas for different hybrid logics. It is highly parameterized to obtain tests of different complexity for the different languages. It has been extensively used in the development of HyLoRes (→ 6.7.5) and HTab (→ 6.7.6).

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

Further reading

- Areces, C. and Heguiabehere, J. *hGen: A Random CNF Formula Generator for Hybrid Languages*. In Methods for Modalities 3 (M4M-3), Nancy, France, September 2003.
- Site and source: http://code.google.com/p/intohylo/

6.7.8 Sparkle

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

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

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

Further reading

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

6.7.9 Haskabelle

Report by:	Florian Haftmann
Status:	working

Since Haskell is a pure language, reasoning about equational semantics of Haskell programs is conceptually simple. To facilitate machine-aided verification of Haskell programs further, we have developed a converter from Haskell source files to Isabelle theory files: Haskabelle.

Isabelle itself is a generic proof assistant. It allows mathematical formulas to be expressed in a formal language and provides tools for proving those formulas in a logical calculus. One such formal language is higher-order logic, a typed logic close to functional programming languages. This is used as translation target of Haskabelle.

Both Haskabelle and Isabelle in combination allow to formally reason about Haskell programs, particularly verifying partial correctness.

The conversion employed by Haskabelle covers only a subset of Haskell, mainly since the higher-order logic of Isabelle has a more restrictive type system than Haskell. A simple adaption mechanisms allows to tailor the conversion process to specific needs.

So far, Haskabelle is working, but there is little experience for its application in practice. Suggestions and feedback welcome.

Further reading

http://isabelle.in.tum.de/haskabelle.html and http://isabelle.in.tum.de/

6.8 Modeling and Analysis

6.8.1 Streaming Component Combinators

Report by:	Blažević Mario
Status:	experimental, actively developed

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

There are two ways to use SCC: as an embedded language in Haskell, or as a set of commands in a

command-line shell. The latter provides its own parser and type checker, but otherwise relies on the former to do the real work.

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

Version 0.3 of SCC has introduced a set of components for processing well-formed XML instances. Current plans for the future work are to improve the parallelism of the streaming combinators and extend the shell scripting language.

The latest stable version of SCC is available from Hackage.

Further reading

- Hackage: http://hackage.haskell.org/cgi-bin/hackage-scripts/package/scc-0.3
- Conference paper: Mario Blažević, Streaming component combinators, Extreme Markup Languages, 2006. http://www.idealliance.org/papers/extreme/proceedings/html/2006/Blazevic01/EML2006Blazevic01.html
- OmniMark implementation: http://developers.omnimark.com/etcetera/streaming-component-combinators.tar.gz

6.8.2 Raskell

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

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

The heart of Raskell is a collection of composable interpreters that support type checking, evaluation, and abstract interpretation of Rosetta specifications. Algebra combinators allow semantic algebras for the same constructs, but for different semantics, to be easily combined. This facilitates further reuse of semantic definitions. Using abstract interpretation, we can transform specifications between semantic domains without sacrificing soundness. This allows for analysis of interactions between two specifications written in different semantic domains. Raskell also includes a Parsec-based Rosetta parser.

The Raskell environment is available for download at the links below. It is continually being updated, so we recommend checking back frequently for updates. To build the Rosetta parser and type checker, you must also install InterpreterLib (→ 5.5.4), available at the third link listed below.

Further reading

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

Contact

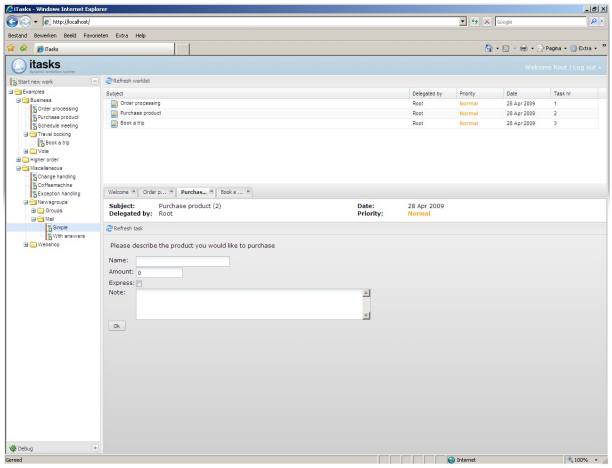
⟨alex@ittc.ku.edu⟩

6.8.3 iTasks

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

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

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



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

Future plans

Currently, we are working on workflow systems in which running workflow processes can be modified dynamically.

Further reading

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

6.8.4 CSP-M Tools at University of Düsseldorf

Report by:	Marc Fontaine
Status:	ongoing

CSP-M is a machine-readable syntax for communicating sequential processes, a formalism invented by Tony Hoare, which is used by several formal-methods-tools.

We are using Haskell to develop tools for parsing, analyzing, type-checking, pretty-printing and animating CSP-M specifications. The ProB model-checker, which is primarily written in Sicstus Prolog, uses a Parsec-based CSP-M parser to read CSP-M syntax.

Further reading

http://www.stups.uni-duesseldorf.de/ProB/overview.php

6.9 Hardware Design

6.9.1 ForSyDe

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

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

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

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

Features

ForSyDe includes two DSL flavors which offer different features:

1. Deep-embedded DSL

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

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

2. Shallow-embedded DSL

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

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

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

Further reading

http://www.ict.kth.se/forsyde/

6.9.2 Lava

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

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

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

Further reading

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

6.9.3 Wired

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

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

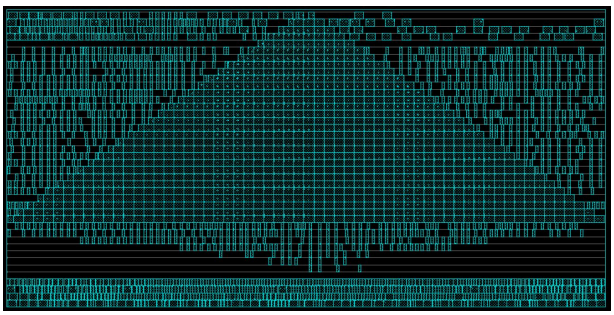
The goal is a system with the following features:

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

We are not very far from this goal. The missing parts are power analysis and support for cell library compilation; and sequential circuits are not yet fully supported.

Wired includes an open-source 45nm standard cell library from Nangate allowing users to play with cutting-edge VLSI technology without the need for any expensive and complicated CAD tools. The system is still quite unstable and has not yet been tested in any larger scale.

The following picture shows the layout of a 32-bit multiplier. Wired was used for generating the netlist and placing the reduction tree.



Further reading

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

6.9.4 Oread

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

The Computer Systems Design Lab is investigating the use of functional languages in the development of

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

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

Further reading

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

Contact

⟨kimmell@ittc.ku.edu⟩

6.10 Natural Language Processing

6.10.1 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. We are in the process of recruiting NLP researchers and users from the Haskell community. In the future, we hope to use the community to discuss libraries and bindings that would be most useful to us and ways of spreading awareness about Haskell in the NLP world.

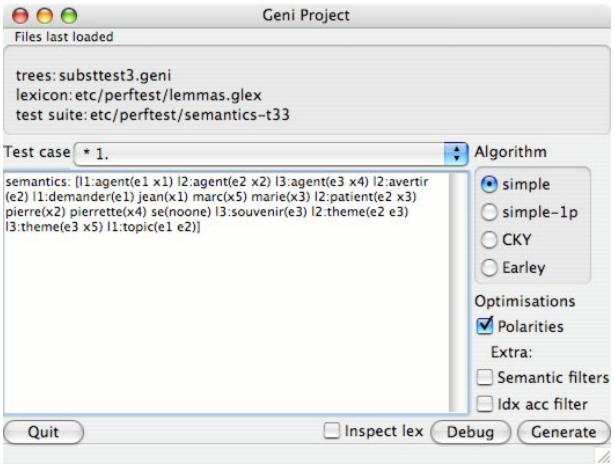
Further reading

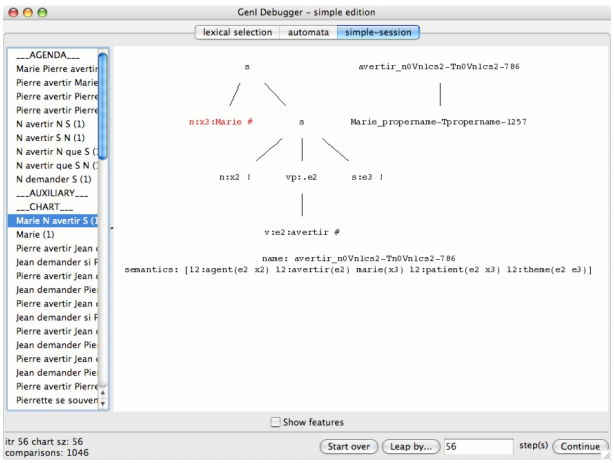
http://projects.haskell.org/nlp

6.10.2 GenI

Report by:	Eric Kow

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





GenI is available on Hackage, and can be installed via cabal-install. Our most recent release of GenI was version 0.17.4, which offers simplified installation with an optional graphical mode, and better help text. For more information, please contact us on the geni-users mailing list.

Further reading

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

6.10.3 Grammatical Framework

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

Grammatical Framework (GF) is a programming language for multilingual grammar applications. It can

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

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

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

Further reading

www.digitalgrammars.com/gf

6.11 Inductive Programming

6.11.1 Inductive Programming

Report by:	Lloyd Allison

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

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

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

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

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

Future plans

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

Further reading

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

6.11.2 IgorII

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

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

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

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

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

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

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

Features

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

Current Status and Future Plans

IgorII is currently still implemented in the reflective rewriting based programming and specification language Maude, and is available on the project page. The porting of IgorII from Maude to Haskell is currently work in progress.

For the future, we plan to move the system to a higher-order context to use morphisms like `map` and `fold` during synthesis, as well as using a polymorphic type system with classes instead of monomorphic types to facilitate the use of background knowledge. Furthermore, the introduction of additional functional constructs (e.g., `let`) and accumulator variables is one of our major goals.

Further reading

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

6.12 Others

6.12.1 Bioinformatics tools

Report by:	Ketil Malde

The Haskell bioinformatics library supports working with nucleotide and protein sequences and associated data. File format support includes sequences in Fasta (with associated quality information), TwoBit, and PHD formats. Recently, support for reading and writing SFF files (from 454-type sequencing) has been added, and also functionality for working with locations.

There is support for sequence alignment tools in the form of parsers for BLAST XML output, Bowtie output, or ACE files. In addition, the standard alignment algorithms (and some non-standard ones) are provided, as well as sequence indexing, complexity calculation, protein translation, etc.

The library is considered in development (meaning things will be added, some functionality may not be as complete or well documented as one would wish, and so on), but central parts should be fairly well documented and come with a QuickCheck test and benchmarking suite.

The library abstracts functionality that is used in a handful of applications, including:

- `flower` — a program for extracting information from and analyzing SFF files.
- `xsact` — an EST clustering program
- `RBR` — a repeat detector/masker
- `clusc` — a tool for calculating cluster similarity with a bunch of metrics
- `dephd` — a sequence quality assessment tool
- `xml2x` — a BLAST postprocessor and GO annotator

Everything is GPLed and available as Darcs repos, at http://malde.org/~ketil/biohaskell/.

6.12.2 Roguestar

Report by:	Christopher Lane Hinson
Status:	early development

Roguestar is a science fiction themed roguelike game written in Haskell. Roguestar uses a client-server model: roguestar-engine is the backend game engine, while roguestar-gl is the OpenGL client.

RSAGL is the RogueStar Animation and Graphics Library, which was written specifically to support roguestar-gl. Every effort has been made to make it accessible to other projects. It includes various levels of support for 3D mathematics, modeling, reactivity and animation, scene composition, and to a limited extent ray tracing.

Roguestar is licensed under the Affero General Public License. RSAGL is licensed under a permissive license.

The 0.4 version is planned for release within the next six months. This should support more combat models, resource gathering and item creation.

Further reading

- http://roguestar.downstairspeople.org
- http://blog.downstairspeople.org

6.12.3 Hpysics

Report by:	Roman Cheplyaka
Status:	experimental

Hpysics is a 3-D physics engine written in Haskell.

At the moment Hpysics supports polyhedral shapes using VClip algorithm for narrow-phase collision detection. A simple OpenGL visualization is included.

Further development plans include, apart from improving engine itself, integration with Grapefruit FRP framework and Lambda-Cube render engine.

Potential users of the physics engine are welcome to guide further development of the project.

The source code is available from public darcs repository under the BSD license.

Further reading

http://haskell.org/haskellwiki/Hpysics

6.12.4 hledger

Report by:	Simon Michael

hledger is a (primarily) command-line accounting tool similar to John Wiegley's "ledger". It reads a plain text journal file describing money or commodity transactions, or timelog entries, and generates precise activity and balance reports.

Since the last report, hledger has reached release 0.4 on Hackage. It has 60 test cases, new features such as basic curses and web-based interfaces, and has had some performance tuning. It is now quite useful for day to day reporting of money and time. Also, the project has a new web address (hledger.org), and has attracted two new committers.

Further reading

http://hledger.org

6.12.5 LQPL — A quantum programming language compiler and emulator

Report by:	Brett G. Giles
Participants:	Dr. J.R.B. Cockett
Status:	in development

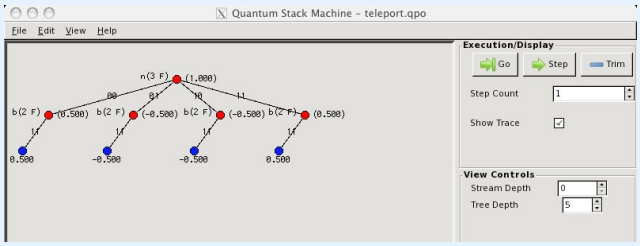
LQPL (Linear Quantum Programming Language) consists of two main pieces, a compiler for a functional quantum programming language and an associated assembler / emulator.

The system was the main subject of the author's master thesis and 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), various predefined unitary transforms, algebraic data types, and operations on purely classical data. The compiler translates LQPL programs into an assembler like language. The emulator, written using Gtk2Hs, translates the assembler to machine code and provides visualization of the program as it executes.

For example, the following procedure implements quantum teleportation:

```
teleport::(n:Qubit,a:Qubit,b:Qubit;b:Qubit) =
{ Not a <= n ;
  Had n;
  measure a of
      |0> => {}  |1> => {Not b};
  measure n of
      |0> => {}  |1> => {RhoZ b}
}
```

The emulator will allow a person to step through this program, displaying the quantum values as a tree. The figure below is a screen shot showing the results after the first measure in teleport.



Since the publication of the thesis, some time and attention has been spent on improving performance and the UI of the emulator.

We plan to release this to the public sometime this year. It will be made available from the first author's website and the programming languages research group's website at the University of Calgary.

Further reading

http://pages.cpsc.ucalgary.ca/~gilesb/research/index.html

6.12.6 Yogurt

Report by:	Martijn van Steenbergen

Yogurt is a MUD client embedded in Haskell. The API allows users to define variables of arbitrary types and hooks that trigger on output from the MUD or input from the user. Other features include timers, multithreading, and logging. Most MUD clients rely on their own custom language; Yogurt, however, relies on Haskell. Even though Yogurt programs are full

Haskell programs, Yogurt is able to dynamically load and reload them using the GHC API, effectively making Yogurt a scripting language.

Ideas for the future include compatibility with `Tintin++` scripts to make migration to Yogurt even more tempting and an expect-like interface for easier interaction with processes.

Further reading

http://code.google.com/p/yogurt-mud/

6.12.7 Dyna 2

Report by:	Wren Ng Thornton
Participants:	Nathaniel W. Filardo, Jason Eisner
Status:	active research

Dyna is a valued-logic programming language with first-class support for dynamic programming. A major goal of the language is to automate many of the common algorithms and optimizations used in natural language parsers and machine translation decoders, making them available for general logic programs.

Starting from Prolog we extend Horn clauses to "Horn equations" by associating each grounding of a rule with a value (not just provability) and aggregating these values to get the value of an item. This extends logic programming with some elements of functional programming, including weighted-logic systems where the form of Horn equations is restricted to a semiring.

My master thesis work was developing a powerful type system which expresses algebraic data types with non-linearity constraints, refinement subtyping, and some aspects of dependent typing. This type system is further enhanced to allow heterogeneous storage of the same semantic type, so that different representations of "the same value" can be used simultaneously in different parts of a program.

Unlike most logic languages, Dyna will have a module system for separating proof universes, which allows multiple programs to share the same RTS instance and allows presenting programs, data sets, and deductive databases with the same API. Dyna will also support agenda-based mixed forward-/backward-chaining inference with memoization and declarative truth maintenance.

Previous work implemented a prototype compiler that generated C++ classes for a restricted form of the language. Currently we are implementing an interpreter in Haskell that covers a broader portion of the language, and are working on the formal underpinnings of these extensions. We intend to use this in the long term as a reference implementation for testing improved algorithms for a next generation Dyna compiler.

Further reading

- Work on the type system and unification algorithms for Dyna 2 can be found in the following papers. These are currently not available online, though refined versions should be available soon.
  - W. Thornton (2008). "Typed Unification in Dyna: An Exploration of the Design Space." Masters Project Report, Johns Hopkins University.
  - W. Thornton (2008). "Heterogeneous Strategies for Unification: Variable–Value Ordering and Optimized Structures." Masters Paper, Johns Hopkins University.
- This paper discusses program transformations for Dyna and gives an early view of the semantics for mixed inference.
  - http://www.cs.jhu.edu/~jason/papers/#fg06
- This paper introduces Dyna 1 where Horn equations are restricted to a semiring.
  - http://cs.jhu.edu/~jason/papers/#emnlp05-dyna
- In association with Dyna there is work on a graphical debugger, called Dynasty, which allows lazy exploration of hypergraphs (representing proof forests).
  - http://cs.jhu.edu/~jason/papers/#infovis06

7 Commercial Users

7.1 Well-Typed LLP

Report by:	Ian Lynagh
Participants:	Björn Bringert, Duncan Coutts

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

We are pleased to have been able to continue to play a role in the development of GHC and the release of 6.10.2, as well as other core parts of the community infrastructure, such as Cabal and the Haskell Platform.

As well as more conventional contracts, we have also been involved in setting up the Industrial Haskell Group (→ 7.9), along with partners including Galois and Amgen. The IHG provides companies with a way to give back to the community, while at the same time benefitting from improvements to the Haskell development platform.

It has been just over a year since we announced the formation of Well-Typed. Looking back, we are pleased with how this first year has gone, and we are excited by the possibilities and challenges that our second year promises to provide. Until next time, happy hacking!

Further reading

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

7.2 SeeReason Partners, LLC

Report by:	Clifford Beshers
Participants:	David Fox, Jeremy Shaw

Clifford Beshers, David Fox, and Jeremy Shaw comprise SeeReason Partners, LLC. We develop web services using Haskell to build our applications. We are currently using AJAX techniques, with JavaScript code generated from Haskell, deployed using HAppS as a web server. We initially planned working with Adobe Flash, based on early work on a Haskell to Flash compiler, but have postponed that work for now.

We have developed and deployed a website for creating art appraisal reports, in use by a private firm. These documents require specific formatting that must be more flexible than simple boilerplate, but for which standard commercial word processing tools proved to be too cumbersome. Multiple users can edit reports simultaneously through a web interface using Wiki markup, which is converted to LaTeX and rendered in PDF format.

We are currently working towards launching AlgebraZam.com, a site to teach mathematics skills, beginning with elementary algebra. The initial launch, expected fourth quarter 2008, will begin with an interactive tool for solving simple algebraic equations.

Formerly core members of the operating systems group at Linspire, Inc., we continue to maintain the tools for managing a Debian Linux distribution that we developed there. Source code for these tools can be found at our public source code repository http://src.seereason.com/. These include a package build system (`autobuilder`) as well as Cabal to Debian conversion tool (`cabal-debian`). We provide current archives of many Haskell packages (including GHC 6.8.3 built with Haddock 2.x) built for recent versions of Debian (unstable) and Ubuntu (8.04 and soon 8.10.) Packages are available at http://deb.seereason.com/. We welcome inquiries from developers interested in using these packages or helping out with continued development.

We can be reached at ⟨(cliff,david,jeremy)@seereason.com⟩ and on `#haskell` respectively as thetallguy, dsfox, and stepcut.

7.3 Credit Suisse Global Modeling and Analytics Group

Report by:	Ganesh Sittampalam

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

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

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

ways to replace or augment the spreadsheet environment.

Our past projects include:

- Adding higher-order functions to Excel, implemented via (Haskell) addin code.
- Tools to transform spreadsheets into directly executable code.
- A "lint" tool to check for common errors in spreadsheets.

Our main project for the last couple of years has been Paradise, a domain-specific language embedded in Haskell for implementing reusable components that can be compiled into multiple target forms. Current backends are Excel spreadsheets and .NET components based on either Winforms or WPF.

A number of modelers have been exposed directly to Haskell by using Paradise, and they have generally picked it up fairly quickly. All new recruits are introduced to Haskell as part of our internal training program.

Our main focus at the moment is the automatic generation of Paradise models for a particular financial product, starting from an algebraic datatype that defines the product. Modelers can override parts of the automatically generated model with a hand-crafted Paradise component if they choose to, providing a good trade-off between speed of development and "beautiful" results.

Further reading

- CUFP 2006 talk about Credit Suisse: http://cufp.galois.com/slides/2006/HowardMansell.pdf
- ICFP 2008 experience report about Paradise: http://www.earth.li/~ganesh/research/paradise-icfp08/paper.pdf http://www.earth.li/~ganesh/research/paradise-icfp08/talk.pdf

7.4 Bluespec tools for design of complex chips

Report by:	Rishiyur Nikhil
Status:	commercial product

Bluespec, Inc. provides a language, BSV, which is being used for all aspects of ASIC and FPGA system design — specification, synthesis, modeling, and verification. All hardware behavior is expressed using *rewrite rules* (Guarded Atomic Actions). BSV borrows many ideas from Haskell — algebraic types, polymorphism, type classes (overloading), and higher-order functions. Strong static checking extends into correct expression of multiple clock domains, and to gated clocks for power management. BSV is universal, accommodating the diverse range of blocks found in SoCs, from algorithmic "datapath" blocks to complex control blocks such as processors, DMAs, interconnects and caches.

Bluespec's core tool synthesizes (compiles) BSV into high-quality RTL (Verilog), which can be further synthesized into netlists for ASICs and FPGAs using other commercial tools. Automatic synthesis from atomic transactions enables design-by-refinement, where an initial executable approximate design is systematically transformed into a quality implementation by successively adding functionality and architectural detail. Other products include fast BSV simulation and development tools. Bluespec also uses Haskell to implement its tools (well over 100K lines of Haskell).

This industrial strength tool has enabled some large designs (over a million gates) and significant architecture research projects in academia and industry. This kind of research was previously feasible only in software simulation. BSV permits the same convenience of expression as SW languages, and its synthesizability further allows execution on FPGA platforms at three orders of magnitude greater speeds, making it possible now to study realistic scenarios.

Status and availability

BSV tools, available since 2004, are in use by several major semiconductor companies and universities. The tools are free for academic teaching and research.

Recent news (last 6 months): (1) Much new infrastructure and libraries to move computation kernels easily onto commodity FPGA boards, for greater speed and/or lower energy; (2) a strange loop, where one customer is applying the capability 1 to a computation kernel written in Haskell; and (3) development of PAClib (Pipeline Architecture Combinators) that make extensive use of higher-order functions to describe DSP algorithms succinctly and with powerful architectural parameterization, exceeding the capabilities of tools that synthesize hardware from C codes.

Further reading

- R.S.Nikhil, *Bluespec, a General-Purpose Approach to High-Level Synthesis Based on Parallel Atomic Transactions*, in *High Level Synthesis: from Algorithm to Digital Circuit, Philippe Coussy and Adam Morawiec (editors)*, Springer, 2008, pp. 129-146.
- Small illustrative examples: http://www.bluespec.com/wiki/SmallExamples
- Winning entry in MEMOCODE 2008 design contest: http://rijndael.ece.vt.edu/memocontest08/
- MIT courseware, "Complex Digital Systems": http://csg.csail.mit.edu/6.375

- A fun example with many functional-programming features — BluDACu, a parameterized Bluespec hardware implementation of Sudoku: http://www.bluespec.com/products/BluDACu.htm

7.5 Galois, Inc.

Report by:	Andy Adams-Moran

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

Galois develops software under contract, and every project (bar three) that we have ever done has used Haskell. The exceptions used ACL2, Poly-ML, SML-NJ, and OCaml, respectively, so functional programming languages and formal methods are clearly our "secret sauce". We deliver applications and tools to clients in industry and the U.S. government. Some diverse examples: Cryptol, a domain-specific language for cryptography (with an interpreter and a compiler, with multiple targets, including FPGAs); a GUI debugger for a specialized microprocessor; a specialized, high assurance, cross-domain web and file server, and Wiki for use in secure environments, and numerous smaller research projects that focus on taking cutting-edge ideas from the programming language and formal methods community and applying them to real world problems.

Web-based technologies are increasingly important to our clients, and we believe Haskell has a key role to play in the production of reliable, secure web software. The culture of correctness Haskell encourages is ideally suited to web programming, where issues of security, authentication, privacy, and protection of resources abound. In particular, Haskell's type system makes possible strong static guarantees about access to resources, critical to building reliable web applications.

To help push further the adoption of Haskell in the domain of web programming, Galois released a suite of Haskell libraries, including:

- json: Support for JavaScript Object Notation
- xml: A simple, lightweight XML parser/generator.
- utf8-string: A UTF8 layer for IO and Strings.
- selenium: Communicate with a Selenium Remote Control server.
- curl: libcurl is a rich client-side URL transfer library.
- sqlite: Haskell binding to sqlite3 databases.
- feed: Interfacing with RSS and Atom feeds
- mime: Haskell support for working with MIME types.

Continuing our deep involvement in the Haskell community, Galois was happy to sponsor the two Haskell hackathons held in the past year, Hac 07 II, in Freiburg, Germany, and Hac4 in Gothenburg, Sweden. Galois also sponsored the second BarCamp Portland, held in early May 2008.

Further reading

http://www.galois.com/.

7.6 IVU Traffic Technologies AG Rostering Group

Report by:	Michael Marte
Status:	released

The rostering group at IVU Traffic Technologies AG has been using Haskell to check rosters for compliance with the "EC Regulation No 561/2006 on the harmonization of certain social legislation relating to road transport" which "lays down rules on driving times, breaks and rest periods for drivers engaged in the carriage of goods and passengers by road".

By reduction from SEQUENCING WITH RELEASE TIMES AND DEADLINES (Garey & Johnson, Computers & Tractability, 1977), it is easy to show that EC 561/2006 is NP complete due to combinatorial rest-time compensation rules.

Our implementation is based on an embedded DSL to combine the regulation's single rules into a solver that not only decides on instances but, in the case of a faulty roster, finds an interpretation of the roster that is "favorable" in the sense that the error messages it entails are "helpful" in leading the dispatcher to the resolution of the issue at hand.

Our EC 561/2006 solver comprises about 1700 lines of Haskell code (including about 250 lines for the C API), is compiled to a DLL with ghc, and linked dynamically into C++ and Java applications. The solver is both reliable (due to strong static typing and referential transparency — we have not experienced a failure in three years) and efficient (due to constraint propagation, a custom search strategy, and lazy evaluation).

Our EC 561/2006 component is part of the IVU.crew software suite and as such is in wide-spread use all over Europe, both in planning and dispatch. So the next time you enter a regional bus, chances are that the driver's roster was checked by Haskell.

Further reading

- EC 561/2006 at EurLex
- The IVU.suite for public transport

7.7 Tupil

Report by:	Chris Eidhof
Participants:	Eelco Lempsink

(,) tupil

Tupil builds reliable web software with Haskell. Using Haskell's powerful ways of abstraction, we feel we can develop even faster than with dynamic scripting languages but with the safety and performance of a language that is statically checked and compiled.

In the last year we were able to successfully use Haskell for different projects: high score web services, music mashups, a payment system for a client and more. It would not have been possible without the vast amount of packages that are available for Haskell these days.

Further reading

- http://tupil.com
- http://blog.tupil.com

7.8 Aflexi Content Delivery Network (CDN)

Report by:	Kim-Ee Yeoh

The Aflexi Content Delivery Network (CDN) is a confederated solution to cost-effective content delivery for publishers, value-added capacity right-sizing for webhosting providers, and a more responsive Internet experience for end-users.

Key elements of the Aflexi CDN comprise a server-side software package that webhosting providers can license to CDN-enable their servers and hosted websites, and a marketplace where a provider can federate with other providers to expand its CDN footprint. Our motto is "Unifying Capacity."

At the heart of the platform is the ability for publishers to transparently combine Aflexi-enabled providers and migrate among a publisher-selected subset. Competition between providers ensures that publishers get a market-efficient rate for delivery bandwidth by making informed decisions based on platform metrics of providers' Quality of Service. By trading excess capacity with other providers, they in turn benefit from the disruptive innovation in capacity recalibration and additional revenue streams afforded by the Aflexi CDN platform.

Aflexi uses Haskell for critical components of its back-end infrastructure. Haskell allows us to rapidly prototype our software efforts via its rich store of open-source libraries on Hackage. Supported by a set of composable concurrency abstractions built on fast lightweight threads, our Haskell code sports more resilient fail-safe features and higher performance while at the same time employing fewer lines of code that ultimately translate to fewer bugs.

Other Haskell projects in development include a domain-specific language (DSL) with termination guarantees (à la *Total Functional Programming*). The DSL furnishes a framework for describing the policies governing content redirection.

Status and availability

The Aflexi CDN platform pre-launched at the start of 2009.

Further reading

http://aflexi.net/

7.9 Industrial Haskell Group

Report by:	Ian Lynagh

The Industrial Haskell Group (IHG) is an organization to support the needs of commercial users of Haskell. Currently it has three partners, including Galois (→ 7.5) and Amgen.

The first activity of the IHG is a collaborative development scheme. Our partners are pooling their resources to fund work on the Haskell development platform to their mutual benefit. The scheme is just getting underway, but has already started producing patches to improve the state of shared library support in GHC.

This initial scheme will run until the end of August. During this time we will also be looking at other ways in which the IHG can help commercial Haskell users, and how the collaborative development scheme can be extended in the future.

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

Further reading

http://industry.haskell.org/

8 Research and User Groups

8.1 Functional Programming Lab at the University of Nottingham

Report by: Liyang HU

The School of Computer Science at the University of Nottingham has recently formed the *Functional Programming Laboratory*, a new research group focused on all theoretical and practical aspects of functional programming, together with related topics such as type theory, category theory, and quantum programming.

The laboratory is led jointly by Thorsten Altenkirch and Graham Hutton, with Henrik Nilsson and Venanzio Capretta as academic staff. With 4 more research staff and some 10 PhD students in our group, we have a wide spectrum of interests:

Containers

Nottingham has been home to the EPSRC grant on *containers*, a semantic model of functional data structures. Thorsten Altenkirch, Peter Hancock, Peter Morris, and Rawle Prince are working with containers to both write and reason about programs. Peter Morris has recently finished his PhD, which used containers as a basis for generic programming with dependent types.

Dependently Typed Programming (DTP)

Peter Morris and Nicolas Oury are working on Epigram, while Nils Anders Danielsson is involved in the development of Agda (→ 3.2.2). Our interests lie both in the pragmatics of using DTP, as witnessed by work on libraries and tools, and in foundational theory, including the Observational Type Theory underlying Epigram 2 and James Chapman's work on normalization. DTP is also used to control and reason about effects, and a number of us are using Agda as a proof assistant to verify programs or programming language theory.

Functional Reactive Programming (FRP)

The FRP team are working on FRP-like and FRP-inspired declarative, domain-specific languages. Under Henrik Nilsson's supervision, Neil Sculthorpe is working on a new, scalable FRP language based on reactive components with multiple inputs and outputs, while George Giorgidze is applying the advantages of FRP to non-causal modeling with the aim of designing a new, more expressive and flexible language for non-causal, hybrid modeling and simulation (→ 6.5.3). Tom Nielsen is implementing a declarative language for experiments, simulations, and analysis in neuroscience. A theme that permeates our work is implementation through embedding in typed functional languages such as Haskell or Agda (→ 3.2.2). The team also maintains Yampa, the latest Haskell-based implementation of FRP.

Quantum Programming

Thorsten Altenkirch and Alexander S Green have been working on the Quantum IO Monad (QIO), an interface from Haskell to Quantum Programming. Taking advantage of abstractions available in Haskell we can provide QIO implementations of many well-known quantum algorithms, including Shor's factorization algorithm. The implementation also provides a constructive semantics of quantum programming in the form of a simulator for such QIO computations.

Reasoning About Effects

Graham Hutton and Andy Gill recently formalized the worker/wrapper transformation for improving the performance of functional programs. Wouter Swierstra and Thorsten Altenkirch have produced functional specifications of the IO monad, as described in Wouter's forthcoming PhD thesis. Mauro Jaskelioff developed a new monad transformer library for Haskell, which provides a uniform approach to lifting operations. Diana Fulger and Graham Hutton are investigating equational reasoning about various forms of effectful programs. Liyang HU and Graham Hutton are working on verifying the correctness of compilers for concurrent functional languages, including a model implementation of software transactional memory.

Teaching

Haskell plays an important role in the undergraduate program at Nottingham, as well as our China and Malaysia campuses. Modules on offer include Functional Programming, Advanced FP, Mathematics for CS, Foundations of Programming, Compilers, and Computer-Aided Formal Verification, among others.

Events

The FP Lab plays a leading role in the Midlands Graduate School in the Foundations of Computing Science, the British Colloquium for Theoretical Computer Science, and the Fun in the Afternoon seminar series on functional programming.

FP Lunch

Every Friday, we gather for lunch with helpings of informal, impromptu-style whiteboard discussions on recent developments, problems, or projects. Summaries of our weekly meetings can be found on the frequently cited *FP Lunch blog*, giving a lively picture of ongoing research at Nottingham.

Later in the afternoon, there is usually a formal hour-long seminar. We are always keen on speakers in any related areas: do get in touch with Thorsten Altenkirch ⟨txa@cs.nott.ac.uk⟩ if you would like to visit. See you there!

8.2 Artificial Intelligence and Software Technology at Goethe-University Frankfurt

Report by:	David Sabel
Participants:	Manfred Schmidt-Schauß

Deterministic calculi. We proved correctness of strictness analysis using abstract reduction, which was implemented by Nöcker for Clean (→ 3.2.3), and by Schütz for Haskell. Our proof is based on the operational semantics of an extended call-by-need lambda calculus which models a core language of Haskell. Furthermore, we proved equivalence of the call-by-name and call-by-need semantics of an extended lambda calculus with `letrec`, `case`, and constructors. Most recently, we investigated and developed proof techniques for showing contextual equivalence in a polymorphically typed `letrec`-calculus.

Nondeterministic calculi. We explored several nondeterministic extensions of call-by-need lambda calculi and their applications. We analyzed a model for a lazy functional language with direct-call I/O providing a semantics for `unsafePerformIO`-calls in Haskell. We investigated a call-by-need lambda-calculus extended by parallel-or and its applications as a hardware description language. We analyzed a call-by-need lambda calculus extended with McCarthy's `amb` and an abstract machine for lazy evaluation of concurrent computations. For all these investigations an emphasis of our research lies in proving program equivalences based on contextual equivalence for showing correctness of program transformations.

Simulation-based proof techniques. We have shown that the soundness proof (w.r.t. contextual equivalence) for mutual similarity of *Matthias Mann* scales up to a class of untyped higher-order nondeterministic call-by-need lambda calculi. For nondeterministic call-by-need calculi with `letrec`, known approaches to prove such a result are inapplicable. In collaboration with *Elena Machkasova* we obtained correctness of a variation of simulation for checking contextual equivalence in an extended non-deterministic call-by-need lambda-calculus with `letrec`. Ongoing research is to adapt and extend the methods to an appropriately defined simulation, and to investigate an extension of the methods to a combination of may- and must-convergence.

Concurrency primitives. Recently, we analyzed the expressivity of concurrency primitives in various functional languages. In collaboration with *Jan Schwinghammer* and *Joachim Niehren*, we showed how to encode Haskell's MVars into a lambda calculus with storage and futures which is an idealized core language of Alice ML. We proved correctness of the encoding using operational semantics and the notions of adequacy and full-abstractness of translations. In her final year thesis *Martina Willig* analyzed the encoding of other concurrency abstractions and implemented them in the caf library using Concurrent Haskell.

Further reading

http://www.ki.informatik.uni-frankfurt.de/research/HCAR.html

8.3 Functional Programming at the University of Kent

Report by:	Olaf Chitil

We are a group of staff and students with shared interests in functional programming. While our work is not limited to Haskell — in particular our interest in Erlang has been growing — Haskell provides a major focus and common language for teaching and research. For autumn 2009 we are seeking PhD students for funded research projects.

Our members pursue a variety of Haskell-related projects, some of which are reported in other sections of this report. Two PhD students recently had their vivas, two new student joined us in the last 6 months. Recently Olaf Chitil updated Heat and released it outside the department. Heat is a deliberately simple IDE for teaching Haskell that has been used at Kent for over three years. Keith Hanna is continuing work on Vital, a document-centered programming environment for Haskell, and on Pivotal, a GHC-based implementation of a similar environment. The Kent Systems Research Group is developing an occam compiler in Haskell (Tock). Neil Brown has created a Haskell library ("Communicating Haskell Processes") based on the Communicating Sequential Processes calculus.

Further reading

- FP group: http://www.cs.kent.ac.uk/research/groups/tcs/fp/

- Refactoring Functional Programs: http://www.cs.kent.ac.uk/projects/refactor-fp/
- Tracing and debugging with Hat: http://www.haskell.org/hat
- Heat: http://www.cs.kent.ac.uk/projects/heat/
- Vital: http://www.cs.kent.ac.uk/projects/vital/
- Pivotal: http://www.cs.kent.ac.uk/projects/pivotal/
- Tock: https://www.cs.kent.ac.uk/research/groups/sys/wiki/Tock
- CHP http://www.cs.kent.ac.uk/projects/ofa/chp/

8.4 Foundations and Methods Group at Trinity College Dublin

Report by:	Andrew Butterfield
Participants:	Glenn Strong, Hugh Gibbons, Edsko de Vries

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

- Supporting OO-Design technique for functional programmers
- Using functional programs as invariants in imperative programming
- Developing a GUI-based 2nd-order equational theorem prover (→ 6.7.3)
- New approaches to uniqueness typing, applicable to Hindley-Milner style type-inferencing
- Equational reasoning for Concurrent Haskell (new)

We have also managed to introduce a new elective course in functional programming at TCD which will be based on the "Real World Haskell" textbook.

Further reading

https://www.cs.tcd.ie/research_groups/fmg/

8.5 Formal Methods at DFKI Bremen and University of Bremen

Report by:	Christian Maeder
Participants:	Mihai Codescu, Dominik Lücke, Christoph Lüth, Christian Maeder, Till Mossakowski, Lutz Schröder

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

We are using the Glasgow Haskell Compiler and many of its extensions to develop the Heterogeneous tool set (Hets). Hets consists of parsers, static analyzers, and proof tools for languages from the CASL family, such as CASL itself, HasCASL, CoCASL, CspCASL, and ModalCASL, and additionally OMDoc and Haskell (via Programatica). The Hets implementation is also based on some old Haskell sources such as bindings to uDrawGraph (formerly Davinci) and Tcl/TK that we maintain.

HasCASL is a general-purpose higher order language which is in particular suited for the specification and development of functional programs; Hets also contains a translation from an executable HasCASL subset to Haskell. There is a prototypical translation of a subset of Haskell to Isabelle/HOL and HOLCF.

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/

8.6 SCIence project

Report by:	Kevin Hammond
Status:	ongoing 5-year project, started in 2006

SCIEnce is a 3.2M euros project that brings together major developers of symbolic computing systems, including GAP, KANT, Maple, and MuPAD, and with the world-leading Centre for Research in Symbolic Computation at RISC-Linz (Austria), OpenMath experts from the Technical University of Eindhoven (Netherlands), and functional programming experts in the Heriot-Watt University (Edinburgh, Scotland) and the University of St Andrews (Scotland).

Our research activity — *"Symbolic computing on the Grid"* — makes essential use of functional programming technology in the form of the GRID-GUM functional programming system for the Grid, which is built on the Glasgow Haskell Compiler. The objective is

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

Our results in this direction are reflected in more than 30 publications and a number of research talks and presentations, listed on the project's website. The public downloads are now under revision and the updated version should appear soon.

Further reading

http://www.symbolic-computation.org/

8.7 Haskell at K.U.Leuven, Belgium

Report by:	Tom Schrijvers
Participants:	Pieter Wuille

We are a two-man unit of functional programming research within the Declarative Languages and Artificial Intelligence group at the Katholieke Universiteit Leuven, Belgium.

Our main project centers around the *Monadic Constraint Programming* (MCP) framework. An initial article on the MCP framework by Tom Schrijvers, Peter Stuckey and Phil Wadler is available. It explains how the framework captures the generic aspects of Constraint Programming in Haskell. Of particular interest is the solver-independent framework for compositional search strategies.

Currently we are extending the framework to act as a modeling language for both the problem description and the search component. The model in Haskell serves as a high-level front-end for a state-of-the-art Constraint Programming system such as Gecode (C++) or Prolog (BProlog, ECL^{i}PS^e). At the same time, the model is also executable using our native Haskell constraint solver.

Our other Haskell-related projects are:

- *Type Checking:* Recent 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 extensive type system, and adding new extensions. This is joint work with Martin Sulzmann, Simon Peyton Jones, Manuel Chakravarty, Dimitrios Vytiniotis, Stefan Monnier and Louis-Julien Guillemette.
- *Test Generation:* Together with master student Timmy Weytjens I am looking at constraint-based generation of test cases. The constraint-based approach should be a more effective than related random (QuickCheck), exhaustive (SmallCheck) or semi/pseudo-constraint-based (LazySmallCheck / narrowing-based) approaches for certain applications.

Further reading

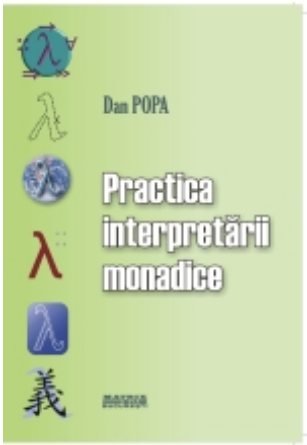
- http://hackage.haskell.org/cgi-bin/hackage-scripts/package/monadiccp
- http://www.cs.kuleuven.be/~toms/Haskell/
- https://www.cs.kuleuven.be/~pieterw/site/Research/Papers/

8.8 Haskell in Romania

Report by:	Dan Popa

This is to report some activities of the Ro/Haskell group. The Ro/Haskell page becomes more and more known. We hope to pass the barrier of the 20000th click on the main page this mounth. The numbers of students and teachers interested in Haskell is increasing. Students begin to have projects using Haskell in order to pass the License Exams. But it is just a beginning. Interests in Data Base Programing with Haskell are growing. (A surprise!).

A book previously published by Mihai Gontineac was released as a free resource. A new book, "The Practice Of Monadic Interpretation" by Dan Popa has been published in November 2008.



The book has a nice foreword written by Simon P.J. and is sharing the experience of a year of interpreter building (2006). It is intended as a student's and teacher's guide to the practical approach of monadic interpretation. The book will probably be used during this academic year in 2–4 universities across Romania (in Iasi, Bacau, Cluj-Napoca).

Haskell products like Rodin (a small DSL a bit like C but written in Romanian) begin to spread, proving the power of the Haskell language. The Pseudocode Language Rodin is used as a tool for teaching basics of computer science in some high-schools from various cities. Some teachers from a high school have requested training concerning Rodin. Rodin was asked to become a FOSS (Free & Open Source Software).

A group of researchers from the field of linguistics located at the State Univ. from Bacau (The LOGOS Group) is declaring the intention of bridging the gap between semiotics, high level linguistics, structuralism, nonverbal communication, dance semiotics (and some other intercultural subjects) AND Computational Linguistics (meaning Pragmatics, Semantics, Syntax, Lexicology, etc.) using Haskell as a tool for real projects. Probably the situation from Romania is not well known: Romania is probably one of those countries where computational linguistics is studied by computer scientists less than linguists.

At Bacau State University, we have teaching Haskell on both Faculties: Sciences (The Computers Science being included) and we hope we will work with Haskell with the TI students from the Fac. Of Engineering, where a course on Formal Languages was requested. "An Introduction to Haskell by Examples" had traveled to The Transilvania Univ. (Brasov) and we are expecting Haskell to be used there, too, during this academic year. Other libraries had received manuals and even donations (in books, of course). Editors seem to be interested by the Ro/Haskell movement, and some of them have already declared the intention of helping us by investing capital in the Haskell books production. A well known Publishing House (MatrixRom) asked us to be the Official Publishing House of the Ro/haskell Group.

Dan Popa is reporting a new technology in order to build the Modular Abstract Syntax Tree of a language processor without Maybe, without Catamorphisms and without Haskell extensions. A paper is available.

There are some unsolved problems, too: PhD. Advisors (specialized in monads, languages engineering, and Haskell) are almost impossible to find. This fact seems to block somehow the hiring of good specialists in Haskell. There was even a funny case when somebody hired to teach Haskell was tested and interviewed by a LISP teacher. Of course, the exam was more or less about lists.

Further reading

- Ro/Haskell: http://www.haskell.org/haskellwiki/Ro/Haskell
- Rodin: http://www.haskell.org/haskellwiki/Rodin

8.9 Assorted Small Portland State University Haskell Bits

Report by:	Bart Massey
Participants:	Julian Kongslie, Jamey Sharp
Status:	mostly under development

Portland State University is a center of development of things like GHC, the House Haskell operating system, etc. Some really amazing work has been done by that group. That group is not us.

Over the past 1.5 years my students and I have participated in a number of Haskell related coding and community activities. Our goals include:

- Better learning and understanding the Haskell programming language.
- Supporting the project work of myself and my group.
- Contributing back to the Haskell community.
- Fun.

Specifically, we have:

- Taken over hosting for the Haskell Sequence / Haskell Weekly News. After some serious initial hiccups, that all seems to be going smoothly.
- Constructed several interesting standalone Haskell applications.
  - Jamey Sharp several years ago constructed a Haskell 802.11 implementation for the Ettus USRP as a Google Summer of Code project.
  - Julian Kongslie has recently completed the first revision of a novel Haskell embedded DSL for Hardware Description, Chortl.
  - Julian Kongslie has recently completed the first revision of a Seaside-style Haskell CGI session framework for web hosting, Riviera.
  - I have mostly completed a Haskell spelling-word suggestion program, thimk, utilizing phonetic codes and edit distance.
- Designed Haskell-supporting hardware. Julian Kongslie has written Verilog implementations of the Haskell G-machine. These are not currently available, as he continues to develop them.
- Constructed library code, and contributed some of it to Hackage and to the Haskell libraries.
  - I have written a command-line argument parsing library, ParseArgs, that I have used in a number of programs. This is in Hackage. However, plans are in the works to convert one of my students' alternate designs currently implemented in C to Haskell, and contribute that instead.

  - I have written a WAVE audio file processing library, and some simple command-line programs for audio processing. These are mostly in Hackage, although I need to package a new release.
  - I have written a PBM graphics file processing library. I am currently working on extending it to support PNG, at which time I will contribute it to Hackage.
  - We have looked at taking over the HaskellDSP signal processing library, after receiving permission from the author to do so. However, this work is currently stalled.
  - In association with my spelling word suggestion application thimk mentioned earlier, I am developing a library of phonetic coding algorithms for contribution to Hackage.
  - I have contributed one patch to the Haskell libraries, to enforce strictness on some sequence constructors as required by the Haskell Report. I also should shortly get my nubOrd function into the libraries.

- Support Haskell in the classroom.
  - Perhaps 20% of my students' projects over the last few years have been submitted in Haskell.
  - I am currently working on tutorial curriculum based on *Real-World Haskell* and the Project Euler problems. Mehana Bisquera-Chang is piloting this curriculum.

- Participated in our local Haskell and open source community. I have given Haskell tutorials at several open source events, such as Portland Bar Camp 2008. Julian and I have both presented our work on several occasions at meetings of the PDX Functional Programming Users' Group.

We would like to thank all of those who have assisted us in this work, but it is hard to enumerate such a large list. Josh Triplett has helped with a number of these projects, and helped me to learn Haskell. Mark Jones and other Faculty at PSU have occasionally helped us out. Don Stewart and others in the Haskell community have provided guidance and support.

What we have been doing is pretty *ad hoc* and disorganized, but we think we have been making real contributions to the Haskell community. Almost all of our work is available to the public under open source licenses — we welcome its use and help with its development.

Further reading

http://wiki.cs.pdx.edu/bartforge

8.10 fp-syd: Functional Programming in Sydney, Australia.

Report by: Ben Lippmeier
Participants: Erik de Castro Lopo

We are a seminar and social group for people in Sydney, Australia interested in Functional Programming and related fields. We meet on the third Thursday of each month and regularly get 25–30 attendees, with a 70/30 industry/research split. Talks this year have included "Intro to PLT Scheme", "A Haskell library for chart plotting", and "Program extraction in a theorem prover like Coq (or Isabelle)". 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