Haskell Communities and Activities Report

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

Eighteenth Edition — May 2010

Janis Voigtländer (ed.)

Robin Adams, Krasimir Angelov, Heinrich Apfelmus
Jim Apple, Dmitry Astapov, Justin Bailey
Doug Beardsley, Jean-Philippe Bernardy, Tobias Bexelius
Edwin Brady, Gwern Branwen, Joachim Breitner
Roman Cheplyaka, Adam Chlipala, Olaf Chitil
Jan Christiansen, Alberto Gómez Corona, Duncan Coutts
Simon Cranshaw, Jácome Cunha, Nils Anders Danielsson
Larry Diehl, Atze Dijkstra, Facundo Dominguez
Jonas Duregård, Marc Fontaine, Patai Gergely
Brett G. Giles, Andy Gill, George Giorgidze
Dmitry Golubovsky, Carlos Gomez, Torsten Grust
Jurriaan Hage, Bastiaan Heeren, Claude Heiland-Allen
Jeroen Janssen, Florian Haftmann, David Himmelstrup
Guillaume Hoffmann, Martin Hofmann, Csaba Hruska
Paul Hudak, Jasper Van der Jeugt, Farid Karimipour
Oleg Kiselyov, Lennart Kolmodin, Michal Konečný
Lyle Kopnicky, Eric Kow, Sean Leather
Bas Lijnse, Andres Löh, Tom Lokhorst
Rita Loogen, Ian Lynagh, John MacFarlane
Christian Maeder, José Pedro Magalhães, Ketil Malde
Arie Middelkoop, Ivan Lazar Miljenovic, Neil Mitchell
Dino Morelli, Matthew Naylor, Jürgen Nicklisch-Franken
Rishiyur Nikhil, Thomas van Noort, Johan Nordlander
Miguel Pagano, Jens Petersen, Simon Peyton Jones
Jason Reich, Matthias Reisner, Stephen Roantree
Fred Ross, Alberto Ruiz, David Sabel
Antti Salonen, Ingo Sander, Uwe Schmidt
Martijn Schrage, Tom Schrijvers, Jeremy Shaw
Axel Simon, Michael Snoyman, Martijn van Steenbergen
Martin Sulzmann, Doaitse Swierstra, Henning Thielemann
Simon Thompson, Wren Ng Thornton, Jared Updike
Marcos Viera, Sebastiaan Visser, Janis Voigtländer
Jan Vornberger, Gregory D. Weber, Stefan Wehr
Mark Wotton, Kazu Yamamoto, Brent Yorgey

Preface

This is the 18th edition of the Haskell Communities and Activities Report. Lots of interesting new stuff; web development has become a hot topic with much activity in particular.

As usual, fresh entries are formatted using a blue background, while updated entries have a header with a blue background. I have continued the strategy, as in the last edition, of replacing with online pointers to previous versions those entries for which I received a liveness ping, but which have seen no essential update for a while. Other entries on which no new activity has been reported for a year or longer have been dropped completely. Please do revive such entries next time if you do have news on them.

A call for new entries and updates to existing ones will be issued on the usual mailing lists in October. Now enjoy the current report and see what other Haskellers have been up to lately. Any kind of feedback is of course very welcome. Specifically, I have been trying to improve the generation of the html version of the report (see http://haskell.org/communities/05-2010/html/report.html), so any remarks on that output could be helpful.

There is no prize question this time. ☺

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

Contents

1 Information Sources

1.1 The Monad.Reader

Report by: Brent Yorgey

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 has been one new issue, featuring articles on space profiling, underappreciated monads, defining monads operationally, and STM. The next issue will be published in May.

Further reading

http://themonadreader.wordpress.com/

1.2 Haskell Wikibook

Report by: Heinrich Apfelmus
Participants: Duplode, Orzetto, David House, Eric Kow, 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. In particular, the Wikibook aims to answer all those conceptual questions that are frequently asked on the Haskell mailing lists.

Everyone including you, dear reader, are invited to contribute, be it by spotting mistakes and asking for clarifications or by ruthlessly rewriting existing material and penning new chapters.

Recent additions include a gentle introduction to generalized algebraic data types (GADTs).

Further reading

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

1.3 Oleg's Mini tutorials and assorted small projects

Report by: Oleg Kiselyov

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

Optimal symbolic differentiation

We demonstrate symbolic differentiation of a wide class of numeric functions without imposing any interpretive overhead. The functions to differentiate can be given to us in separately compiled modules, with no available source code. We produce a (compiled, if needed) function that is an exact, *algebraically simplified analytic derivative* of the given function. Our approach is an application of normalization-by-evaluation. To avoid interpretive overhead, we rely on Template Haskell (if the interpretive overhead is acceptable, Template Haskell can be avoided).

Our approach also produces higher- and partial derivatives. Currently we support algebraic functions and a bit of trigonometry.

http://okmij.org/ftp/Computation/Computation/Generative.html#diff-th

Logic programming in Haskell optimized for reasoning

We demonstrate an executable model of the evaluation of definite logic programs, i.e., of resolving Horn clauses presented in the form of definitional trees. Our implementation, DefinitionTree, is yet another embedding of Prolog in Haskell. It is distinguished not by speed or convenience. Rather, it is explicitly designed to formalize evaluation strategies such as SLD and SLD-interleaving, to be easier to reason about and so help prove termination and other properties of the evaluation strategies. The main difference of DefinitionTree from other embeddings of Prolog in Haskell is

the absence of name-generation effects. We need neither `gensym` nor the state monad to ensure the unique naming of logic variables. Since naming and evaluation are fully separated, the evaluation strategy is no longer concerned with fresh name generation and so is easier to reason about equationally. We have indeed used DefinitionTree to prove basic properties of solution sets obtained by SLD or SLD-resolution strategies.

http://okmij.org/ftp/Haskell/misc.html#reasoned-LP

Choosing a type-class instance based on the context

This mini-tutorial, written together with Simon Peyton-Jones, explains how to overload operations based not on the type of an expression but on the class to which an expression's type belongs. For example, we want to define an overloaded operation `print` to be equivalent to `(putStrLn . show)` when applied to showable expressions, whose types are the members of the class `Show`. For other types, the operation `print` should do something different (e.g., print that no show function is available, or, for Typeable expressions, write their type instead). The problem is not trivial because normally the type-checker selects an instance of the type-class based only on the instance head. The instance constraints are not taken into account during the selection process. The trick is to re-write a constraint `C a` which succeeds or fails, into a predicate constraint `C' a flag`, which always succeeds, but once discharged, unifies `flag` with a type-level Boolean `HTrue` or `HFalse`.

http://okmij.org/ftp/Haskell/types.html#class-based-overloading

1.4 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 literate source file. Spanish and Japanese translations are also available.

Further reading

http://cheatsheet.codeslower.com

1.5 Practice of Functional Programming

Report by:	Dmitry Astapov
Status:	five issues ready, collecting materials for issue #6

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

The magazine attempts to keep a bi-monthly release schedule, with Issue #6 slated for release in June 2010.

Full contents of current and past issues are available in PDF from the official site of the magazine free of charge.

Articles are in Russian, with English annotations.

Further reading

http://fprog.ru/ for issues ##1–5

1.6 Cartesian Closed Comic

Report by:	Roman Cheplyaka
Participants:	Maria Kovalyova

Cartesian Closed Comic, or CCC, is a webcomic about Haskell, the Haskell community, and anything else related to Haskell. It is published irregularly. The comic is sometimes inspired by "Quotes of the week" published in Haskell Weekly News. New strips are posted to the Haskell reddit and Planet Haskell. The archives are also available.

Further reading

http://ro-che.info/ccc/

2 Implementations

2.1 The Glasgow Haskell Compiler

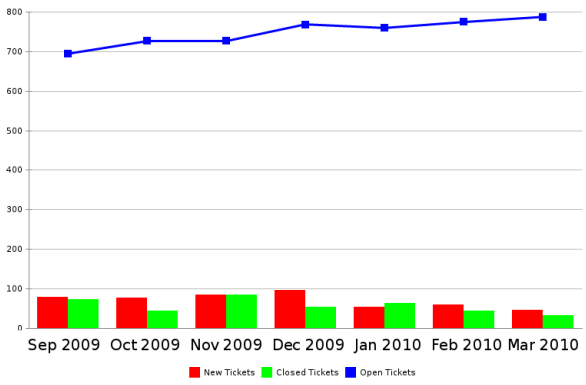
Report by:	Simon Peyton Jones
Participants:	many others

In the past 6 months we have made the first 2 releases from the 6.12 branch. 6.12.1 came out in December, while 6.12.2 was released in April. The 6.12.2 release fixes many bugs relative to 6.12.1, including closing 81 trac tickets. For full release notes, and to download it, see the GHC webpage (http://www.haskell.org/ghc/download_ghc_6_12_2.html). We plan to do one more release from this branch before creating a new 6.14 stable branch.

GHC 6.12.2 will also be included in the upcoming Haskell Platform release (→ 5.2). The Haskell platform is the recommended way for end users to get a Haskell development environment.

Ongoing work

Meanwhile, in the HEAD, the last 6 months have seen more than 1000 patches pushed from more than a dozen contributors. As the following graph shows, tickets are still being opened faster than we can close them, with the total open tickets growing from around 700 to almost 800. We will be looking in the near future at improving the effectiveness of the way we use the bug tracker.



Language changes

We have made only a few small language improvements. The most significant ones concern quasi-quotations, implementing suggestions from Kathleen Fisher:

- Quasi-quotes can now appear as a top-level declaration, or in a type, as well as in a pattern or expression.
- Quasi-quotes have a less noisy syntax (no "$").

Here is an example that illustrates both:

```
f x = x+1
[pads| ...blah..blah... |]
```

The second declaration uses the quasi-quoter called `pads` (which must be in scope) to parse the "`...blah..blah..`", and return a list of Template Haskell declarations, which are then spliced into the program in place of the quote.

Type system

Type families remain the hottest bit of GHC's type system. Simon PJ has been advertising for some months that he intends to completely rewrite the constraint solver, which forms the heart of the type inference engine, and that remains the plan although he is being slow about it. The existing constraint solver works surprisingly well, but we have lots of tickets demonstrating bugs in corner cases. An upcoming epic (70-page) JFP paper "Modular type inference with local assumptions" brings together all the key ideas; watch Simon's home page.

The mighty simplifier

One of GHC's most crucial optimizers is the Simplifier, which is responsible for many local transformations, plus applying inlining and rewrite-rules. Over time it had become apparent that the implementation of INLINE pragmas was not very robust: small changes in the source code, or small wibbles in earlier optimizations, could mean that something with an INLINE pragma was not inlined when it should be, or vice versa.

Simon PJ therefore completely re-engineered the way INLINE pragmas are handled:

- GHC now takes a "snapshot" of the original RHS of a function with an INLINE pragma.
- The function is now optimized as normal, but when the function is inlined it is the snapshot, not the current RHS, that is inlined.
- The function is inlined only when it is applied to as many arguments as the LHS of its original definition. Consider

```
f1, f2 :: Int -> Int -> Int
{-# INLINE f1 #-}
f1 x = \y -> <blah>
{-# INLINE f2 #-}
f2 x y = <blah>
```

Here `f1` will be inlined when it is applied to one argument, but `f2` will only be inlined if it appears applied to two arguments. This turns out to be helpful in reducing gratuitous code bloat.

Another important related change is this. Consider

```
{-# RULE "foo" flip (flop x) = <blah> #-}
test x = flip y ++ flip y
  where
    y = flop x
```

GHC will not fire rule "foo" because it is scared about duplicating the redex `(flop x)`. However, if you declare that `flop` is CONLIKE, thus

```
{-# NOINLINE [1] CONLIKE flop #-}
```

this declares that an application of `flop` is cheap enough that even a shared application can participate in a rule application. The CONLIKE pragma is a modifier on a NOINLINE (or INLINE) pragma, because it really only makes sense to match `flop` on the LHS of a rule if you know that `flop` is not going to be inlined before the rule has a chance to fire.

The back end

GHC's back end has been a ferment of activity. In particular,

- David Terei made a LLVM back end for GHC (http://hackage.haskell.org/trac/ghc/wiki/Commentary/Compiler/Backends/LLVM). It is not part of the HEAD, but we earnestly hope that it will become so.

- John Dias, Norman Ramsey, and Simon PJ made a lot of progress on Hoopl, our new representation for control flow graphs, and accompanying functions for dataflow analysis and transformation. There is a paper (Norman Ramsey, John Dias, and Simon Peyton Jones, Hoopl: A Modular, Reusable Library for Dataflow Analysis and Transformation, submitted to ICFP'10; http://research.microsoft.com/en-us/um/people/simonpj/papers/c--/), and Hoopl itself is now a standalone, re-usable Cabal package, which makes it much easier for others to use.

The downside is that the code base is in a state of serious flux:

- We still have two back-end pipelines, because we do not trust the new one to drop the old one. See: http://hackage.haskell.org/trac/ghc/wiki/Commentary/Compiler/NewCodeGen.

- We are in the midst of pushing the new Hoopl into GHC.

Runtime system work (SimonM)

There has been a lot of restructuring in the RTS over the past few months, particularly in the area of parallel execution. The biggest change is to the way "black-holes" work: these arise when one thread is evaluating a lazy computation (a "thunk"), and another thread or threads demands the value of the same thunk. Previously, all threads waiting for the result of a thunk were kept in a single global queue, which was traversed regularly. This led to two performance problems. Firstly, traversing the queue is $O(n)$ in the number of blocked threads, and we recently encountered some benchmarks in which this was the bottleneck. Secondly, there could be a delay between completing a computation and waking up the threads that were blocked waiting for it. Fortunately, we found a design that solves both of these problems, while adding very little overhead.

We also fixed another pathological performance case: when a large numbers of threads are blocked on an MVar and become unreachable at the same time, reaping all these threads was an $O(n^2)$ operation. A new representation for the queue of threads blocked on an MVar solved this problem.

At the same time, we rearchitected large parts of the RTS to move from algorithms involving shared data structures and locking to a message-passing style. As things get more complex in the parallel RTS, using message-passing lets us simplify some of the invariants and move towards having less shared state between the CPUs, which will improve scaling in the long run.

The GC has seen some work too: the goal here is to enable each processor ("capability" in the internal terminology) to collect its private heap independently of the other processors. It turns out that this is quite tricky to achieve in the context of the current architecture, but we have made some progress in this direction by privatizing more of the global state and simplifying the GC data structures by removing the concept of "steps", while keeping a simple aging policy, which is what steps gave us previously.

Data Parallel Haskell

In the last months, our focus has been on improving the scalability of the Quickhull benchmark, and this work is still ongoing. In addition, Roman has invested significant energy into the increasingly popular vector package and the NoSlow array benchmark framework. Package vector is our next-gen sequential array library, and we will replace the current sequential array component (dph-prim-seq) with package vector sometime in the next few months.

We completed a first release of the regular, multi-dimensional array library introduced in the previous status report. The library is called Repa and is available from Hackage (http://hackage.haskell.org/package/repa). The library supports shape-

polymorphism and works with both the sequential and parallel DPH base library. We discuss the use and implementation of Repa in a draft paper (Gabriele Keller, Manuel M. T. Chakravarty, Roman Leshchinskiy, Simon Peyton Jones, and Ben Lippmeier: Regular, shape-polymorphic, parallel arrays in Haskell, submitted to ICFP'10; http://www.cse.unsw.edu.au/~chak/papers/KCLPL10.html). We have shown that Repa can produce efficient and scalable code for FFT and relaxation algorithms and would be very interested to hear from early adopters who are willing to try Repa out in an application they care about. At the start of the year, Ben Lippmeier has joined the project. He has started to improve our benchmarks infrastructure and worked on Repa.

Other miscellaneous stuff

- GHC makes heavy use of sets and finite maps. Up till now it has used its own home-grown `UniqFM` and `FiniteMap` modules. Milan Straka (visiting as an intern from the Czech Republic) has:
  - made GHC use the `containers` package instead, which happily makes compilation go a few percent faster;
  - developed some improvements to `containers` that makes it go faster still.

  So `UniqFM` and `FiniteMap` are finally dead. Hurrah for Hackage!
- The Threadscope tool for visualizing parallel execution was released. The tool is ripe for improvement in many ways; if you are interested in helping, let us know.

Nightly builds

For some time, it has been clear to us that Buildbot is not the perfect tool for our nightly builds. The main problem is that it is very susceptible to network wibbles, which means that many of our builds fail due to a network issue mid-build. Also, any customization beyond that anticipated by the configuration options provided requires some messy python coding, poking around inside the buildbot classes. Additionally, we would like to implement a "validate-this" feature, where developers can request that a set of patches is validated on multiple platforms before being pushed. We could not see an easy way to do this with buildbot.

When the darcs.haskell.org hardware was upgraded, rather than installing buildbot on the new machine, we made the decision to implement a system that better matched our needs instead. The core implementation is now complete, and we have several machines using it for nightly builds.

We are always keen to add more build slaves; please see http://hackage.haskell.org/trac/ghc/wiki/Builder if you are interested. Likewise, patches for missing features are welcome! The (Haskell) code is available at http://darcs.haskell.org/builder/.

2.2 LHC

Report by:	David Himmelstrup
Participants:	Austin Seipp
Status:	active development

LHC is a backend for the Glorious Glasgow Haskell Compiler (→ 2.1), adding low-level, whole-program optimization to the system. It is based on Urban Boquist's GRIN language, and using GHC as a frontend, we get most of its great extensions and features.

Essentially, LHC uses the GHC API to convert programs to external core format — it then parses the external core, and links all the necessary modules together into a whole program for optimization. We currently have our own base library (heavily and graciously taken from GHC). This base library is similar to GHC's (module-names and all), and it is compiled by LHC into external core and the package is stored for when it is needed. This also means that if you can output GHC's external core format, then you can use LHC as a backend.

The short-term goal is to make LHC faster, easier to use, and more complete in its coverage of Haskell 98.

Further reading

- http://lhc.seize.it/
- http://lhc-compiler.blogspot.com/

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 (→ 5.3.2)).

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

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

UHC, what is new? UHC is the Utrecht Haskell Compiler, supporting almost all Haskell98 features and most of Haskell2010, plus experimental extensions. After the first release of UHC in spring 2009 we have been working on the next release, which we expect to have available this summer. Although UHC did start its life as a compiler for research and experimentation, much of the recent work has focussed on improving and stabilizing UHC for actual use. The highlights of the next release will be:

- Support for building libraries with Cabal.
- A base library sufficient for Haskell98.
- Support for most of the Haskell2010 language features.
- A new garbage collector, replacing the Boehm GC we have been using until recently.
- More stable implementation of both compiler and runtime, with many bugfixes.

All of the above is already available for download from the UHC svn repository.

UHC, what do we currently do? As part of the UHC project, the following (student) projects and other activities are underway (in arbitrary order):

- Jan Rochel: "Realising Optimal Sharing", based on work by Vincent van Oostrum and Clemens Grabmayer.
- Tom Lokhorst: type based static analyses.
- Jeroen Leeuwestein: incrementalization of whole program analysis.
- Atze van der Ploeg: lazy closures.
- Paul van der Ende: garbage collection & LLVM.
- Arie Middelkoop (& Lucília Camarão de Figueiredo): type system formalization and automatic generation from type rules.
- Jeroen Fokker: GRIN backend, whole program analysis.
- Călin Juravle: base libraries.
- Levin Fritz: base libraries for Java backend.
- Andres Löh: Cabal support.
- José Pedro Magalhães: generic deriving.
- Doaitse Swierstra: parser combinator library.
- Atze Dijkstra: overall architecture, type system, bytecode interpreter backend, garbage collector.

Some of the projects are highlighted directly below.

Type based static analysis (Tom Lokhorst) We are working on various static optimization transformations on top of the recently introduced typed core intermediate language. A particular focus is optimizing code based on the results of a type based strictness analysis (Stefan Holdermans and Jurriaan Hage, Making "Stricterness" More Relevant, PEPM '10). We are currently investigating several approaches to optimizing higher order functions that are polymorphic in their strictness properties.

Lazy closures (Atze van der Ploeg) We are investigating cheaper ways to construct closures by reusing information already present in frames (incarnation records). In this scheme a frame may be used by a closure after the frame's function has ended so we put frames on the heap instead of the stack. If a frame's function has ended, the frame may contain more information than is necessary for the closures that use it, the garbage collector needs to be aware of this so that we do not save too much.

Garbage collection & LLVM (Paul van der Ende) We want to extend the LLVM backend of UHC with accurate garbage collection. The LLVM compiler is known to do various aggressive transformations that might break static stack descriptors. We will exploit the existing shadow-stack functionality of the LLVM framework to connect it with the garbage collection library.

Generic deriving (José Pedro Magalhães) Recently we wanted to extend the **deriving** support in UHC to allow deriving for other common type classes (such as *Functor* and *Typeable*, for example). However, instead of hard-wiring particular classes in the compiler, we decided to allow the user to specify how instances should

be derived for any type class, using simple generic programming techniques. Currently we are working on implementing this new feature and providing **deriving** support for a number of useful classes.

Background UHC actually is a series of compilers of which the last is UHC, plus infrastructure for facilitating experimentation and extension:

- The implementation of UHC is organized as a series of increasingly complex steps, and (independent of these steps) a set of aspects, thus addressing the inherent complexity of a compiler. Executable compilers can be generated from combinations of the above.
- The description of the compiler uses code fragments which are retrieved from the source code of the compilers, thus keeping description and source code synchronized.
- Most of the compiler is described by UUAG, the Utrecht University Attribute Grammar system (→ 4.1.1), thus providing a more flexible means of tree programming.

For more information, see the references provided.

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 Haskell front end for the Clean compiler

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

We are currently working on a front end for the Clean compiler (→ 3.2.4) 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

2.6 The Reduceron

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

Over the past year, work on the Reduceron has continued, and we have reached our goal of improving runtime performance by a factor of six! This has been achieved through many small improvements, spanning architectural, runtime, and compiler-level advances.

Two main by-products have emerged from the work. First, *York Lava*, now available from Hackage, is the HDL we use. It is very similar to Chalmers Lava, but supports a greater variety of primitive components, behavioral description, number-parameterized types, and a first attempt at a Lava prelude. Second, *F-lite* is our subset of Haskell, with its own lightweight toolset.

There remain some avenues for exploration. We have taken a step towards parallel reduction in the form of *speculative evaluation of primitive redexes*, but have not yet attempted the *Reducera* — multiple Reducerons running in parallel. And recently, Jason has been continuing his work on the F-lite supercompiler (→ 4.1.4), which is now producing some really nice results.

Alas, the time to take stock and publish a full account of what we have already done is rapidly approaching!

Further reading

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

2.7 Platforms

2.7.1 Haskell in Gentoo Linux

Report by:	Lennart Kolmodin

Gentoo Linux currently officially supports GHC 6.10.4, including the latest Haskell Platform (→ 5.2) for x86, amd64, sparc, and ppc64. For previous GHC versions we also have binaries available for alpha, hppa and ia64.

The full list of packages available through the official repository can be viewed at http://packages.gentoo.org/category/dev-haskell?full_cat.

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

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

We have also recently started an official Gentoo Haskell blog where we can communicate with our users what we are doing http://gentoohaskell.wordpress.com/.

There is also an overlay which contains more than 300 extra unofficial and testing 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 with minimal user intervention. Notable packages in the overlay include the latest version of the Haskell Platform as well as the latest 6.12.2 release of GHC, as well as popular Haskell packages such as pandoc (→ 6.4.1) and gitit (→ 6.3.3).

More information about the Gentoo Haskell Overlay can be found at http://haskell.org/haskellwiki/Gentoo. Using Darcs (→ 6.1.1), it is easy to keep up to date, to submit new packages, and to fix any problems in existing packages. It is also available via the Gentoo overlay manager "layman". If you choose to use the overlay, then any problems should be reported on IRC (`#gentoo-haskell` on freenode), where we coordinate development, or via email ⟨haskell@gentoo.org⟩ (as we have more people with the ability to fix the overlay packages that are contactable in the IRC channel than via the bug tracker).

Through recent efforts we have developed a tool called "haskell-updater" http://www.haskell.org/haskellwiki/Gentoo#haskell-updater (initiated by Ivan Lazar Miljenovic). This is a replacement of the old `ghc-updater` script for rebuilding packages when a new version of GHC is installed which is now not only written in Haskell but will also rebuild broken packages. "haskell-updater" is still in active development to further refine and add to its features and capabilities.

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

2.7.2 Fedora Haskell SIG

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

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

We have been updating packages for Fedora 13 due to ship soon with ghc-6.12.1 with shared libraries enabled, and haskell-platform-2010.1.0.0: special thanks to Rakesh Pandit for reviewing 11 new packages formerly in ghc extralibs. Darcs is updated to 2.4 (thanks for Conrad Meyer and Lorenzo Villani for reviewing new dependent packages). New packaging macros in ghc-rpm-macros have removed nearly all the remaining tedium of packaging libraries for Fedora with simpler .spec file templates in the Fedora cabal2spec package.

Fedora 14 changes will probably be more modest: likely 6.12.2 and more libraries and programs from hackage.

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

Further reading

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

3 Language

3.1 Extensions of Haskell

3.1.1 Eden

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

The parallel Eden runtime environment for GHC 6.8.3 is available from the Marburg group on request. The Eden extension of GHC 6.12 will soon be released. Support for Glasgow parallel Haskell (GpH, http://haskell.org/communities/11-2008/html/report.html#sect3.1.2) 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. A first parallel Haskell Hackathon has taken place in St Andrews from December 10th till 12th, 2009. It has been a lively event triggering various activities to develop the common parallel runtime environment further.

Parallel program executions can be visualized using the Eden trace viewer tool EdenTV. Recent results show that the Eden system behaves equally well on workstation clusters and on multi-core machines.

Recent and Forthcoming Publications

- Oleg Lobachev and Rita Loogen: *Estimating Parallel Performance, a Skeleton-based Approach*, Technical Report No 2010–2, Department of Mathematics and Computer Science, Philipps-Universität Marburg, 2010.
- Mischa Dieterle, Thomas Horstmeyer, Rita Loogen: *Skeleton Composition Using Remote Data*, in: Practical Aspects of Declarative Programming 2010 (PADL'10), LNCS 6009, Springer 2010, 337–353.
- Thomas Horstmeyer, Rita Loogen: *Grace — Graph-based Communication in Eden*, Trends in Functional Programming, Volume 10, Intellect 2010, 1–16.
- Mustafa Aswad, Phil Trinder, Abdallah Al Zain, Greg Michaelson, Jost Berthold: *Low Pain vs No Pain Multi-core Haskells*, Trends in Functional Programming, Volume 10, Intellect 2010, 49–64.
- Lidia Sánchez-Gil, Mercedes Hidalgo-Herrero, Yolanda Ortega-Mallén: *An Operational Semantics for Distributed Lazy Evaluation*, Trends in Functional Programming, Volume 10, Intellect 2010, 65–80.
- Mercedes Hidalgo-Herrero, Yolanda Ortega-Mallén: *To be or not to be... Lazy (in a Parallel Context)*, Electronic Notes in Theoretical Computer Science (ENTCS) VOL. 258, Elsevier, 2009.

Further reading

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

3.1.2 XHaskell project

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

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

The latest version of the library-based regular expression pattern matching component is available via the google code web site. We are currently working on a paper describing the key ideas of the approach.

3.1.3 HaskellActor

Report by:	Martin Sulzmann
Status:	stable

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 working towards a distributed version of Haskell actor following the approach of Frank Huch, Ulrich Norbisrath: Distributed Programming in Haskell with Ports, IFL'00.

3.1.4 HaskellJoin

Report by:	Martin Sulzmann
Status:	stable

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

In this honors thesis, Olivier Pernet (a student of Susan Eisenbach) provides a nicer monadic interface to the HaskellJoin library.

Further reading

http://sulzmann.blogspot.com/2008/12/parallel-join-patterns-with-guards-and.html

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 compilation techniques and even semantic questions in the area of language extensions like encapsulation and function patterns. Besides activities regarding the language itself, there is also an active development of tools concerning Curry (e.g., the documentation tool CurryDoc, the analysis environment CurryBrowser, the observation debuggers COOSy and iCODE, the debugger B.I.O. (http://www-ps.informatik.uni-kiel.de/currywiki/tools/oracle_debugger), EasyCheck (http://haskell.org/communities/05-2009/html/report.html#sect4.3.2), 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

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

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

New since last time:

- Version 2.2.6 has been released, with experimental support for universe polymorphism.
- FreeBSD users can now install Agda using FreshPorts.

Further reading

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

3.2.3 Idris

Report by:	Edwin Brady
Status:	active development

Idris is an experimental language with full dependent types. Dependent types allow types to be predicated on values, meaning that some aspects of a program's behavior can be specified precisely in the type. The language is closely related to Epigram and Agda (→ 3.2.2). It is available from http://www.idris-lang.org, and there is a tutorial at http://www.cs.st-andrews.ac.uk/~eb/Idris/tutorial.html.

Idris aims to provide a platform for realistic programming with dependent types. By realistic, we mean the ability to interact with the outside world and use primitive types and operations, to make a dependently typed language suitable for systems programming. This includes networking, file handling, concurrency, etc. Idris emphasizes programming over theorem proving, but nevertheless integrates with an interactive theorem prover. It is compiled, via C, and uses the Boehm-Demers-Weiser garbage collector.

One goal of the project is to show that Idris, and dependently typed programming in general, can be efficient enough for the development of real world verified software. To this end, Idris is currently being used to develop a library for verified network protocol implementation, with example applications.

Further reading

http://www.idris-lang.org/

3.2.4 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 of notable features:

- Clean is a lazy, pure, and higher-order functional programming language with explicit graph-rewriting semantics.
- Although Clean is by default a lazy language, one can smoothly turn it into a strict language to obtain optimal time/space behavior: functions can be defined lazy as well as (partially) strict in their arguments; any (recursive) data structure can be defined lazy as well as (partially) strict in any of its arguments.
- Clean is a strongly typed language based on an extension of the well-known Milner/Hindley/Mycroft type inferencing/checking scheme including the common higher-order types, polymorphic types, abstract types, algebraic types, type synonyms, and existentially quantified types.
- 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.

- The Clean type system supports dynamic types, allowing values of arbitrary types to be wrapped in a uniform package and unwrapped via a type annotation at run-time. Using dynamics, code and data can be exchanged between Clean applications in a flexible and type-safe way.
- Clean 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 (→ 2.5) for the future plans.

Further reading

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

3.2.5 Timber

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

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

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 latest release of the Timber compiler system is v 1.0.3 and dates back to May 2009. More recent developments are available in the on-line source code repository, including a new way of organizing and accessing external interfaces, a simplified command syntax, and many bug fixes. A proper release of this version is in the making and will be announced before summer 2010.

The new view of external interfaces separates access to OS, hardware or library services from the definition of a particular run-time system. This move greatly simplifies the construction of both external bindings and cross-compilation targets, which is utilized in ongoing development of Xlib, OpenGL, iPhone as well as ARM7 support. Some of these targets will be part of the upcoming release, while others are scheduled for a follow-up at the end of the year. This later release will also contain a newly developed back-end targeting Javascript and HTML5, with the purpose of making Timber applicable to web programming in a reactive and strongly typed fashion.

Other active projects include interfacing the compiler to memory and execution-time analysis tools, extending it with a supercompilation pass, and taking a fundamental grip on the generation of type error messages. The latter work will be based on principles developed for the Helium compiler (→ 2.3).

Further reading

http::/timber-lang.org

3.2.6 Ur/Web

Report by:	Adam Chlipala
Status:	beta release

Ur/Web is a domain-specific language for building modern web applications. It is built on top of the **Ur** language as a custom standard library with special compiler support. Ur draws inspiration from a number of sources in the world of statically-typed functional programming. From Haskell, Ur takes purity, type classes, and monadic IO. From ML, Ur takes eagerness and a module system with functors and type

abstraction. From the world of dependently-typed programming, Ur takes a rich notion of type-level computation.

The Ur/Web extensions support the core features of today's web applications: "Web 1.0" programming with links and forms, "Web 2.0" programming with non-trivial client-side code, and interaction with SQL database backends. Considering programmer productivity, security, and scalability, Ur/Web has significant advantages over the mainstream web frameworks. Novel facilities for statically-typed metaprogramming enable new styles of abstraction and modularity. The type system guarantees that all kinds of code interpretable by browsers or database servers are treated as richly-typed syntax trees (along the lines of familiar examples of GADTs), rather than as "strings", thwarting code injection attacks. The whole-program optimizing compiler generates fast native code which does not need garbage collection.

The open source toolset is in beta release now and should be usable for real projects. I expect the core feature set to change little in the near future, and the next few releases will probably focus on bug fixes and browser compatibility.

Further reading

http://www.impredicative.com/ur/

4 Tools

4.1 Transforming and Generating

4.1.1 UUAG

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

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

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

Nonterminals (data types), productions (data constructors), attributes, and rules for attributes can be specified separately, and are woven and ordered automatically. Recently, we enhanced these aspect-oriented programming features. It is now possible to add rules that transform a value of a synthesized attribute, or to transform a child and its inherited and synthesized attributes.

The system is in use by a variety of large and small projects, such as the Utrecht Haskell Compiler UHC (→ 2.4), the editor Proxima for structured documents (→ 6.4.5), the Helium compiler (→ 2.3), the Generic Haskell compiler, UUAG itself, and many master student projects. The current version is 0.9.19 (April 2010), is extensively tested, and is available on Hackage.

We are working on the following enhancements of the UUAG system, building on attribute grammar research of the past in a modern setting:

Parallel evaluation We aim to evaluate attributes in parallel to take advantage of current multi-core processors. The static dependencies between attributes allow us to identify parts that are independent and schedule them simultaneously in a safe and efficient way.

Incremental evaluation We generate code that adapts incrementally to changes in the input data. The goal is to reuse the outcome of previous computations, by recomputing only those parts of the computation that are affected by the changes.

Fixpoint evaluation When static dependencies between attributes are circular, results can still be computed through lazy evaluation when the dynamic dependencies are not. We are now incorporating a fixed-point evaluation scheme that computes a result even in case of a dynamic cycle.

Furthermore, we investigate extensions of the AG formalism to make AGs more suitable to express type inferencing. We made prototype implementations that facilitate the dynamic construction of inference trees, to use AGs for bidirectional type rules and constraint solving.

Further reading

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

4.1.2 AspectAG

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

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

Introduction

Attribute Grammars (AGs), a general-purpose formalism for describing recursive computations over data types, avoid the trade-off which arises when building software incrementally: should it be easy to add new data types and data type alternatives or to add new operations on existing data types? However, AGs are usually implemented as a pre-processor, leaving e.g. type checking to later processing phases and making interactive development, proper error reporting and debugging difficult. Embedding AG into Haskell as a combinator library solves these problems. Previous attempts at embedding AGs as a domain-specific language were based on extensible records and thus exploiting Haskell's type system to check the well-formedness of the AG, but fell short in compactness and the possibility to abstract over oft occurring AG patterns. Other attempts used a very generic mapping for which the AG well-formedness could not be statically checked. We present a typed embedding of AG in Haskell satisfying all these requirements. The

key lies in using HList-like typed heterogeneous collections (extensible polymorphic records) and expressing AG well-formedness conditions as type-level predicates (i.e., typeclass constraints). By further type-level programming we can also express common programming patterns, corresponding to the typical use cases of monads such as Reader, Writer, and State. The paper presents a realistic example of type-class-based type-level programming in Haskell.

Background

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

Further reading

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

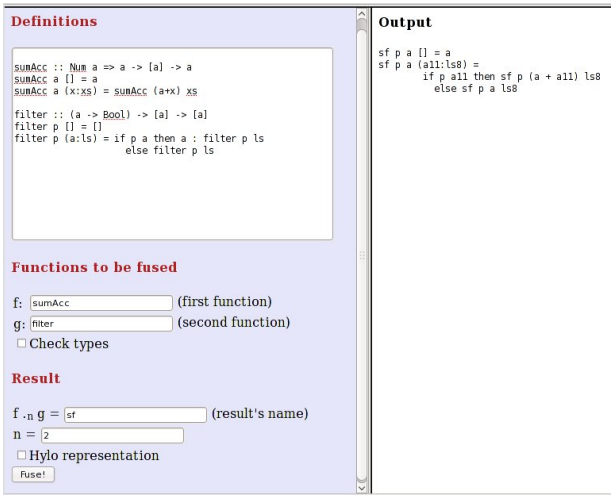
4.1.3 HFusion

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

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

We offer a web interface to test the technique on user-supplied recursive definitions. The user can ask HFusion to transform a composition of two functions into an equivalent program which does not build the intermediate data structure involved in the composition. In future developments of the tool we plan to find fusable compositions within programs automatically.

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



Further reading

- Documentation about the tool can be found in HFusion home
- HFusion web interface is available from this URL

4.1.4 Optimus Prime

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

Optimus Prime is project developing a *supercompiler* for programs written in *F-lite*, the subset of Haskell used by the Reduceron (→ 2.6). It draws heavily on Neil Mitchell's work on the Supero supercompiler for YHC Core.

The project is still at the highly experimental stage but preliminary results are very encouraging. The process appears to produce largely *deforested* programs where higher-order functions have been *specialized*. This, as a consequence, appears to enable further gains from mechanisms such as *speculative evaluation of primitive redexes* on the Reduceron architecture.

Optimus Prime supercompilation has led to a 74% reduction in the number of Reduceron clock-cycles required to execute some micro-examples.

Work continues on improving the execution time of the supercompilation transformation and improving the performance of the supercompiled programs.

Contact

http://www.cs.york.ac.uk/people/?username=jason

Further reading

http://optimusprime.posterous.com/

4.1.5 Derive

Report by:	Neil Mitchell
Status:	v2.3.0

The Derive tool is used to generate formulaic instances for data types. For example given a data type, the Derive tool can generate 34 instances, including the standard ones (Eq, Ord, Enum etc.) and others such as Binary and Functor. Derive can be used with SYB, Template Haskell or as a standalone preprocessor. This tool serves a similar role to DrIFT, but with additional features.

Recently Derive has had many derivations added, including new Uniplate (→ 5.8.1) instances. The mechanism to derive instances by example has been rewritten, and the revised mechanism is described in the associated Approaches and Applications of Inductive Programming 2009 paper.

Further reading

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

4.1.6 Agata

Report by:	Jonas Duregård
Participants:	Koen Claessen
Status:	experimental, active

The Agata library (Agata Generates Algebraic Types Automatically) is an outcome of my master's thesis work at Chalmers University of Technology. The library uses Template Haskell to derive instances of the QuickCheck `Arbitrary` class for (almost) any Haskell data type.

The generators differ from regular QuickCheck generators in that they maintain scalability even for types analogous to nested collection data structures (e.g., `[[[[a]]]]`, where the standard QuickCheck generator tends to generate values that contain millions of `a`'s). Generators also guarantee that independent components of the same type have the same expected size, e.g., in `(a,[a])` the single `a` will have the same expected size as any `a` in the list.

Although a few additional features are to be implemented in the near future, efforts will be focused on documentation and improving performance. When the library is stable and well documented, the possibility of integrating it into the QuickCheck package may be explored.

Further reading

- http://hackage.haskell.org/package/Agata
- Agata — Random generation of test data (Master's thesis), http://gupea.ub.gu.se/bitstream/2077/22087/1/gupea_2077_22087_1.pdf

4.1.7 lhs2TEX

Report by:	Andres Löh
Status:	stable, maintained

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

The program is stable and can take on large documents.

Since version 1.14, lhs2TeX has an experimental mode for typesetting Agda code.

The current version is 1.15. Due to changes in the handling of Unicode in ghc-6.12, this version should be built with ghc-6.10. In the near future, version 1.16 will be released that is hopefully behaving correctly when built with ghc-6.12.

Further reading

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

4.2 Analysis and Profiling

4.2.1 HTF: a test framework for Haskell

Report by:	Stefan Wehr
Status:	beta, active development

The Haskell Test Framework (*HTF* for short) lets you define unit tests, QuickCheck properties, and black box tests in an easy and convenient way. The HTF uses a custom preprocessor that collects test definitions automatically. Furthermore, the preprocessor allows the HTF to report failing test cases with exact file name and line number information.

Initially created in 2005, HTF was not actively developed for almost five years. Development resumed in 2010, adding many improvements to the code base.

Further reading

- http://hackage.haskell.org/package/HTF
- Tutorial: http://www.factisresearch.com/2010/03/htf/

4.2.2 SourceGraph

Report by:	Ivan Lazar Miljenovic
Status:	version 0.6.1.0

SourceGraph is a utility program aimed at helping Haskell programmers visualize their code and perform

simple graph-based analysis (representing entities as nodes in the graphs and function calls as directed edges), which started off as an example of how to use the Graphalyze library (→ 5.7.1), which is designed as a general-purpose graph-theoretic analysis library. These two pieces of software were originally developed as the focus of my mathematical honors thesis, "Graph-Theoretic Analysis of the Relationships Within Discrete Data".

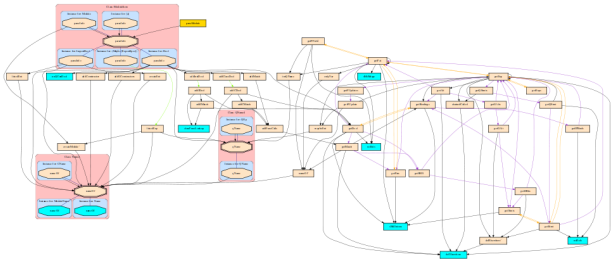
Whilst fully usable, SourceGraph is currently limited in terms of input and output. It analyses all `.hs` and `.lhs` files recursively found in the provided directory, parsing most aspects of Haskell code (cannot parse Haskell code using CPP, HaRP, TH, FFI and XML-based Haskell code; difficulty parsing Data Family instances, unknown modules and record puns and wildcards). The results of the analysis are created in an Html file in a "SourceGraph" subdirectory of the project's root directory.

Various refinements have been implemented since the last release, including:

- "Implicitly exported" entities (e.g., class method instance definitions from external classes) are now supported; support for these is not perfect and may include more entities than it should.
- Addition of depth analysis (based upon how many function calls are needed from an exported entity).
- Better visualizations, including edge categorizations; the generated Dot code is also saved if users wish to tweak these.

Current analysis algorithms utilized include: alternative module groupings, whether a module should be split up, root analysis, depth 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).

Sample SourceGraph analysis reports can be found at http://code.haskell.org/~ivanm/Sample_SourceGraph/SampleReports.html. A tool paper on SourceGraph was presented at the ACM SIGPLAN 2010 Workshop on Partial Evaluation and Program Manipulation.



Further reading

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

4.2.3 HLint

Report by:	Neil Mitchell
Status:	v1.6

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

There have been numerous feature improvements since the last HCAR. HLint supports Unicode and more fully integrates with a C pre-processor. Many hints have been added, some of which were submitted by users. A new mode has been added to hunt for suitable hints given source code. There have been substantial speed improvements.

Further reading

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

4.2.4 A Haskell source file scanning tool

Report by:	Christian Maeder

The Haskell source file scanning tool `scan` is supposed to be a complement for `hlint` (→ 4.2.3). Whereas `hlint` makes suggestions to improve your expressions, `scan` makes suggestions about your source file format regarding white spaces, layout and comments, as usually described by style guides.

The `scan` tool is also able to write back an untabified file without trailing white space, with proper blanks around infix operators and after commas, and with a single final newline.

I use this tool to keep my Haskell sources tidy and reduce mere white space changes in evolving revisions. You are encouraged to do so, too.

Further reading

http://projects.haskell.org/style-scanner/

4.2.5 hp2any

Report by:	Patai Gergely
Status:	experimental, on hold

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

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

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

Further reading

http://www.haskell.org/haskellwiki/Hp2any

4.3 Development

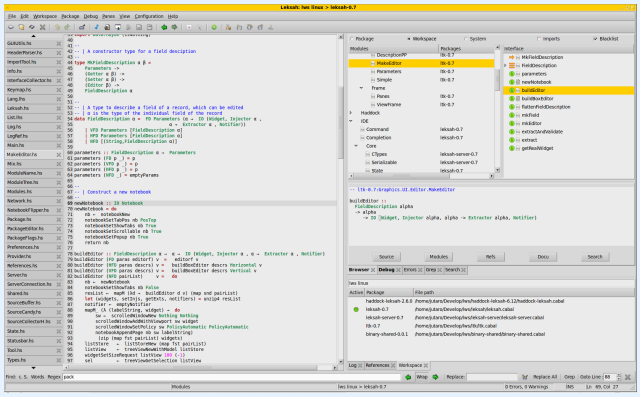
4.3.1 Leksah — Toward a Haskell IDE

Report by:	Jürgen Nicklisch-Franken

Leksah is a Haskell IDE written in Haskell, it uses Gtk+, and runs on Linux, Windows, and Mac OS X. Leksah is intended to be a practical tool to support the Haskell development process. Leksah is completely free.

Some features of Leksah:

- It uses the cabal package format and incorporates a cabal file editor.
- It offers Workspaces for complex projects with multiple packages with automatic build of dependencies.
- It contains a module browser that allows you to find type information about all the functions/symbols available in the packages installed on your system.
- For most packages it shows as well haddock style comments, and gives direct navigation to sources.
- It integrates ghci debugging (including continuous recompilation) that allows you to type check and evaluate highlighted code snippets from within the editor itself. Includes a scratch buffer for testing ideas.
- It includes a helper for automatic addition of import statements.
- Offers a Haskell-customized editor with "source candy".
- Multi-window support for a multi head setting.
- Many standard features of IDEs like: Jump to errors, Auto Completion, Grep integration, ...
- Configurable with session support, keymaps, and flexible appearance.



Future plans

- Enhance usability and fix open bugs for the 1.0 release.
- Concept and implementation of an extension mechanism.
- Better integration of Yi as editor component.

The project needs more users and developers!

Further reading

http://leksah.org/

4.3.2 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 Haskell. Heat provides a small number of supporting features and is easy to use.

Heat is portable, small and works on top of 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.

Over the summer 2009 Heat was completely re-engineered to provide a simple and cleaner internal structure for future development. This new version still misses a few features compared to the current 3.1 version. (Small) modifications for making Heat work with GHC instead of Hugs have also been submitted. A new release is still in the works.

Further reading

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

4.3.3 HaRe — The Haskell Refactorer

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

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

Our project, *Refactoring Functional Programs*, has as its major goal to build a tool to support refactorings in Haskell. The HaRe tool is now in its fifth 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 documentation. Please let us know if you are using the API.

Snapshots of HaRe are available from our webpage, as are related presentations and publications from the group (including LDTA'05, TFP'05, SCAM'06, PEPM'08, PEPM'10, Huiqing's PhD thesis and Chris's PhD thesis). The final report for the project appears there, too.

Chris Brown has presently passed his PhD; his PhD thesis entitled "Tool Support for Refactoring Haskell Programs" is available from our webpage.

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.

Further reading

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

4.3.4 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.

The DarcsWatch service is moved to a new machine, urchin.earth.li, to accommodate its growths. DarcsWatch continues to be used by the xmonad project (→ 6.1.2), the Darcs project itself, and a few developers. At the time of writing, it was tracking 32 repositories and 2631 patches submitted by 171 users.

Further reading

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

4.3.5 DPM — Darcs Patch Manager

Report by:	Stefan Wehr
Participants:	David Leuschner
Status:	beta, active development

The Darcs Patch Manager (*DPM* for short) is a tool that simplifies working with the revision control system darcs (http://darcs.net). It is most effective when used in an environment where developers do not push their patches directly to the main repository but where patches undergo a reviewing process before they are actually applied.

The current feature set of DPM is quite stable. In our company (→ 7.5), we actively use DPM to keep track of all patches sent to various projects. At the Haskell hackathon 2010 in Zürich, we started working on support for tracking conflicts between patches. We did not yet finish this work, but hope to provide a new DPM release with support for conflicts in May 2010.

There is some overlap between DPM and darcswatch (→ 4.3.4). The main difference between darcswatch and DPM is that the former mainly targets developers whereas the latter helps reviewers doing their work.

Further reading

- http://hackage.haskell.org/package/DPM
- Tutorial: http://www.factisresearch.com/2010/03/dpm/

4.3.6 HSFFIG

Report by:	Dmitry Golubovsky
Status:	release

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

HSFFIG has been in development since 2005, and was recently released on Hackage. The current version is 1.1.2 which is mainly a bug-fix release for the version 1.1.

The package provides a small library to link with programs using auto-generated imports, and two executable programs:

- `hsffig`: a filter program which reads pre-processed include files from standard input, and produces one large .hsc file;
- `ffipkg`: a program which automates the process of building a Cabal package out of C include files by the means of automated running hsffig and other tools necessary to build a Haskell package.

Further reading

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

4.3.7 Hubris

Report by:	Mark Wotton
Participants:	James Britt, Larry Diehl, Josh Price, Tatsuhiro Ujihisa, Andrew Grimm
Status:	beta

Hubris is an in-process bridge between Ruby and Haskell, allowing Ruby programs to use Haskell code without writing boilerplate.

It is now easier to install, and some 64 bit bugs have been fixed.

To get it on Linux:

```
cabal install hubris
gem install hubris
```

Mac OS X is a bit harder because support for dynamic libraries has not been merged into the GHC mainline yet, but it is in the pipe. Further plans:

- work with new versions of Ruby without reinstallation of Hubris Haskell-side support code
- support for passing RTS flags to the Haskell process
- translation instance injection (i.e., express equivalents for complex Haskell datatypes in Ruby and vice versa)
- multiple argument support
- some way of storing non-translatable instances on the ruby side — ideally, you should be able to have a Ruby list of Haskell functions, and apply each of them in turn. Currently only translateable data types are marshalled.

Further reading

- http://github.com/mwotton/Hubris-Haskell
- http://github.com/mwotton/Hubris
- http://www.engineyard.com/blog/2010/a-hint-of-hubris/
- http://www.jamesbritt.com/2010/3/13/a-purely-functional-tale-of-a-bridge-compose-of-hubris

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

We had a recent release of Cabal-1.8 and cabal-install-0.8. These are available from hackage and are included with the upcoming major release of the Haskell Platform (→ 5.2). The primary change is that these releases work with GHC 6.12.

There is also a new "cabal init" command to help users create an initial ".cabal" file. This should help users to follow recommended practise, rather than copying old idioms from old packages.

There is also a new feature to maintain an HTML contents page for the haddock documentation for all installed packages.

Looking forward

Matthew Gruen will be working on the new Hackage server implementation for his Google Summer of Code project. The aim is to improve the design so that more features can be added and then to add many of the new features that users so frequently request. We have been discussing using a REST design with the server presenting both an HTML interface for humans and also an interface for automated clients.

As ever, there are many improvements we want to make to Cabal, cabal-install and Hackage. I am pleased to report that we have had a few new contributors in the last few months. Nevertheless, our limiting factor is the amount of volunteer development time and code-review time. The bug tracker is well maintained so it should be relatively clear to new contributors what is in need of attention and which tasks are considered relatively easy.

Further reading

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

5.2 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 1500 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`. As of GHC 6.12 the task of shipping an entire platform has been transferred to the Haskell Platform.

Recent progress

At the time of writing we are about to make the second major release of the platform. This will be the 2010.2.0.x release series. This release series will be based on GHC 6.12.2 (or later compatible point releases). Beta versions of the platform, labelled 2010.1.0.x and using GHC 6.12.1, have been available for some weeks.

While there have been no new packages included in this major release, there have been a few significant upgrades including QuickCheck version 2, the latest versions of the 'regex-*' packages and of course GHC 6.12.x.

Looking forward

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

We would like to invite package authors to propose new packages for future major releases. We

also invite the rest of the community to take part in the review process on the libraries mailing list `libraries@haskell.org`. The procedure involves writing a package proposal and discussing it on the mailing list with the aim of reaching a consensus. Details of the procedure are on the development wiki.

Further reading

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

5.3 Auxiliary Libraries

5.3.1 hmatrix

Report by:	Alberto Ruiz
Status:	stable, maintained

The *hmatrix* library is a purely functional interface to numerical linear algebra, internally implemented using GSL, BLAS, and LAPACK.

Recent work includes changes in the internal data structures to make the package compatible with Roman Leshchinskiy's *vector*. The modules for the GSL special functions have been moved to a separate package (*hmatrix-special*).

Further reading

- http://code.haskell.org/hmatrix
- http://perception.inf.um.es/tensor

5.3.2 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.

5.3.3 mueval

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

Mueval is a code evaluator for Haskell; it employs the GHC API as provided by the Hint library (http://haskell.org/communities/11-2008/html/report.html#hint). 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

Since the last HCAR report, the internals have been cleaned up further, a number of minor bugs squashed, tests added, and mueval updated to avoid bitrot.

We are currently working on the following:

- Refactoring modules to render Mueval more useful as a library
- Removing the POSIX-only requirement
- Merging in Chris Done's mueval-interactive fork, which powers http://tryhaskell.org/

Further reading

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

5.4 Parsing and Transforming

5.4.1 ChristmasTree

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

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

5.4.2 First Class Syntax Macros

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

The idea of having an extensible language is appealing and raises the question how to construct extensible compilers. In recent years we have developed a collection of techniques which together enable this in Haskell: transformation of typed abstract syntax trees makes it possible to construct parsers on the fly in a type-safe way, parser combinators make it possible to construct parsers dynamically, and first-class attribute grammars make it possible to define semantics compositionally.

We use these techniques together in constructing compilers out of a collection of pre-compiled, statically type-checked, possibly mutually dependent "language-definition fragments". This way of constructing a compiler brings syntax macros for free.

Background

The solution we present builds on a couple of related developments:

- the introduction of typed abstract syntax (Arthur I. Baars and S. Doaitse Swierstra: Typing dynamic typing, ICFP'02.)
- the introduction of a naming structure which makes it possible to represent mutually dependent structures and the possibility to manipulate such structures while keeping types correct (http://haskell.org/communities/05-2009/html/report.html#sect5.5.6, http://hackage.haskell.org/package/TTTAS; Arthur I. Baars, S. Doaitse Swierstra, and Marcos Viera: Typed transformations of typed abstract syntax, TLDI'09.)
- the description and composition of typed grammar descriptions (Marcos Viera, S. Doaitse Swierstra, and E. Lempsink: Haskell, do you read me?: constructing and composing efficient top-down parsers at runtime, Haskell '08.)
- the typed Left-Corner Transform which removes left-recursion from a grammar (Arthur I. Baars, s. Doaitse Swierstra, and Marcos Viera: Typed transformations of typed grammars: The left corner transform, LDTA'09.)
- the possibility to construct self-analyzing, error correcting parser on the fly (http://hackage.haskell.org/package/uulib; S. Doaitse Swierstra: Parser combinators: from toys to tools, Haskell'00; S. Doaitse Swierstra: Combinator parsing: A Short Tutorial, LerNet ALFA Summer School 2008)
- the type safe extension of semantics via attribute grammar fragments and their composition, which make attribute grammars first class Haskell values, which can be transformed, composed and finally evaluated ((→ 4.1.2), http://hackage.haskell.org/package/AspectAG; Marcos Viera, S. Doaitse Swierstra, and Wouter Swierstra: Attribute grammars fly first-class: how to do aspect oriented programming in Haskell, ICFP'09.)

Further reading

http://www.cs.uu.nl/wiki/Center/SyntaxMacrosForFree

5.4.3 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. Recently a few modifications were made to the old library in order to deal with recent changes in the Haskell offside rule.

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 has appeared in the LNCS lecture notes (S. Doaitse Swierstra, Combinator Parsers: A Short Tutorial, Language Engineering and Rigorous Software Development 2009, LNCS 5520). The text is also available as a technical report at http://www.cs.uu.nl/research/techreps/UU-CS-2008-044.html. The new library was released as the *uu-parsinglib* library, which has found its place in the *Text.ParserCombinators* category on Hackage.

Features

- Much simpler internals than the old library (http://haskell.org/communities/05-2009/html/report.html#sect5.5.8).
- 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 in-

put 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*.

Future plans

The next version of the library, with an abstract interpretation part in order to get the parsing speed we got used to, will be released on Hackage again, with an extensive documentation of its internals and ways to use them. Since many aspects of the old library, such as its applicative interface and the possibility to build, e.g., a parser for permutation phrases, have now become available elsewhere in other packages, we will also try to make the new library to conform as much as possible with these new developments.

Furthermore we plan to carry the implementation of the Left-Corner Transform to a version of this library, so we will be able to deal with left-recursive grammar here too.

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⟩.

5.4.4 Regular Expression Matching with Partial Derivatives

Report by:	Martin Sulzmann
Participants:	Kenny Zhuo Ming Lu

Regular expression matching is a classical and well-studied problem. Prior work applies DFA and Thompson NFA methods for the construction of the matching automata. We propose the novel use of derivatives and partial derivatives for regular expression matching. We show how to obtain algorithms for various matching policies such as POSIX and greedy left-to-right. Our benchmarking results show that the run-time performance is promising and that our approach can be applied in practice.

Further reading

http://sulzmann.blogspot.com/2010/04/regular-expression-matching-using.html

5.5 Mathematical Objects

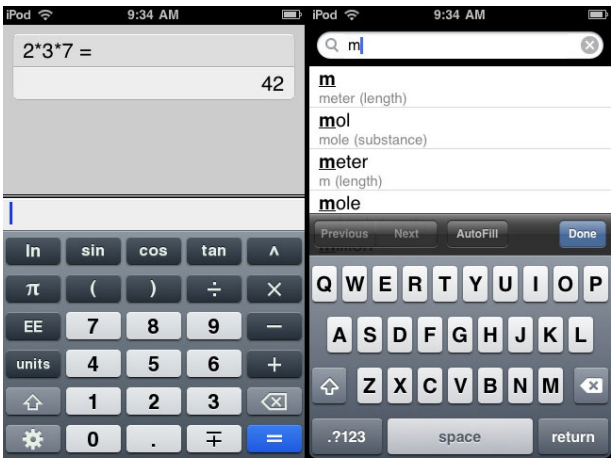
5.5.1 Halculon: units and physical constants database

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.

The example web application now has a mobile version available (tested in iPhone OS 3.1, Safari 3.0, and Firefox 2.0). For best results on the iPhone or iPod touch, Add to Home Screen to use the application in full screen. The calculator works offline, too.



Further reading

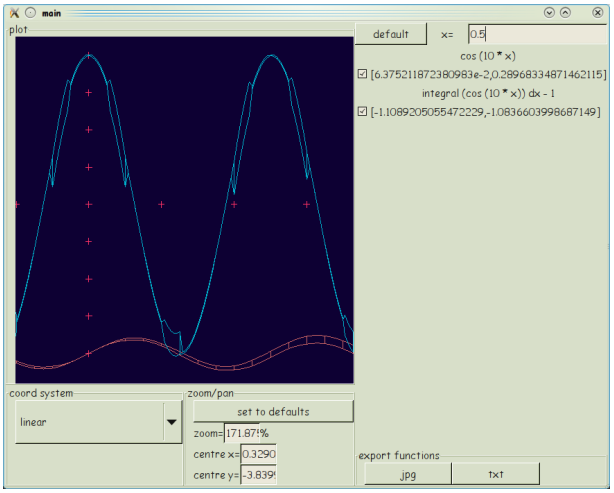
- http://www.updike.org/articles/Units
- http://www.updike.org/halculon/
- http://www.updike.org/halcmobile/

5.5.2 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 safely rounded interval arithmetic with multiple backends (pure Haskell floating point numbers, MPFR, machine doubles) and with support for inner rounding, anti-consistent intervals and Kaucher arithmetic
- AERN-RnToRm: arbitrary precision safely-rounded arithmetic of piece-wise polynomial function enclosures (PFEs) for functions over n-dimensional real intervals with support for inner rounding, anti-consistent intervals and approximated Kaucher arithmetic
- 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)

The current versions have been fairly stable for our purposes.

We are currently redesigning and rewriting the libraries almost from scratch with the following goals:

- A larger number of simpler and more reusable type classes instead of the few and fairly complex type classes provided in the current version; this includes type classes such as RoundedLattice or RoundedMultiplication.
- A more thorough approach to testing, with properties defined alongside the type classes.
- Support for both pure arithmetic and in-place updates using the ST monad for extra efficiency with backends written in C such a MPFR.
- A faster implementation of polynomial arithmetic, probably with a core written in C.

Further reading

- See Haddock documentation via Hackage — has links to research papers.
- New version under construction on http://code.google.com/p/aern/.

5.5.3 logfloat

Report by:	Wren N.G. Thornton
Status:	stable?
Current release:	0.12.1
Portability:	GHC 6.8, GHC 6.10, Hugs Sept2006

See: http://haskell.org/communities/05-2009/html/report.html#sect5.6.7.

5.6 Data types and data structures

5.6.1 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. 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://code.haskell.org/OOHaskell). The HList library relies on common extensions of Haskell 98.

HList is being used in AspectAG (→ 4.1.2), typed EDSL of attribute grammars, and in HaskellDB. There has been many miscellaneous changes related to the names of exposed modules, fixity declarations. Patches by Adam Vogt significantly improve the Haddock-generated documentation. The current version is 0.2.1; it works with GHC 6.12.

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/

5.6.2 Verified priority queues

Report by:	Jim Apple
Status:	stable

A priority queue (or sometimes "heap") is a container supporting the insertion of elements and the extraction of the minimum element. Gerth Brodal and Chris Okasaki presented a purely functional priority queue that also supports an $O(1)$ meld operation in their paper "Optimal Purely Functional Priority Queues". This project provides an implementation of these priority queues that has been verified using the Coq proof assistant.

It is available on Hackage and can be installed with `cabal install meldable-heap`. The Coq proofs are included in the package.

Further reading

http://hackage.haskell.org/package/meldable-heap/

5.6.3 bytestring-trie

Report by:	Wren N.G. Thornton
Status:	slow development
Current release:	0.2.1.1
Portability:	Haskell 98 + CPP

See: http://haskell.org/communities/05-2009/html/report.html#sect5.7.4.

5.7 Data processing

5.7.1 Graphalyze

Report by:	Ivan Lazar Miljenovic
Status:	version 0.9.0.0

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. Changes since previous versions have focused on refining the contents of the library and inclusion of new analysis algorithms, with future plans to re-write the document generation modules to use pretty-printing functions.

Graphalyze is used in SourceGraph (→ 4.2.2) (which is the driving force behind improvements to Graphalyze), and was initially developed as part of my Mathematics Honours' thesis, *Graph Theoretic Analysis of Relationships Within Discrete Data*. The focus of this thesis was to develop computational tools to allow people to analyze discrete data sets.

Further reading

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

5.7.2 Bravo

Report by:	Matthias Reisner
Status:	experimental; active development

Bravo is a general-purpose text template library, providing the parsing and generation of templates at compile time. Templates can be read from strings or files and for each a new record data type is created, allowing convenient access to all template variables in a type-safe manner. The data type creation it achieved by the use of the Template Haskell language extension.

Features

Compared to other template libraries, Bravo's features are:

- Static template processing: All templates are read, parsed, and processed at compile time, so no extra file access or error handling at runtime is necessary.
- Multiple templates per file: Bravo allows the user to define multiple templates per file with arbitrary comments between them, e.g., for template documentation.

- Conditional template evaluation: To check conditions at runtime and return the appropriate template text, conditional template expressions are provided.
- Embedding of Haskell expressions: Bravo allows the user to embed arbitrary Haskell 98 expressions combined with template variables. The set of permitted functions/operators/types can be controlled using Haskell's module system.
- Customized data type generation: Bravo uses a default scheme for the data type creation that can be replaced by a user defined scheme easily.

Future plans

There were plans to extend Bravo's capabilities by introducing new template expressions, e.g., to "map" a template over a list of values. Contrary to expectations, this requires the internal parser to be rewritten and split into lexer and parser. However, this would also improve extensibility and stability of the implementation. Further work will include performance analysis and handling of different input encodings. Support for custom template expression delimiters (the current are `{{` and `}}`) and caching are also planned.

Further reading

- http://www.haskell.org/haskellwiki/Bravo
- http://hackage.haskell.org/package/Bravo

5.8 Generic and Type-Level Programming

5.8.1 uniplate

Report by:	Neil Mitchell

Uniplate is a library for writing simple and concise generic operations. Uniplate has similar goals to the original Scrap Your Boilerplate work, but is substantially simpler and faster. If you are writing any sort of compiler, you should be using a generics library. If you do not know any generics libraries, Uniplate is a good place to start.

Uniplate has recently undergone major revisions. The new version drops Haskell 98 compatibility, in favor of Haskell 2010 compatibility — simplifying the module layout. All the instances have been revised with a focus on performance. Some of the instances can now be generated by the Derive tool (→ 4.1.5). The instances based on the Data class have been optimized and extended — they now work on more types, and run faster.

Further reading

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

5.8.2 Generic Programming at Utrecht University

Report by:	José Pedro Magalhães
Participants:	Stefan Holdermans, Johan Jeuring, Sean Leather, Andres Löh, Thomas van Noort
Status:	actively developed

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

Currently, we are maintaining five generic programming libraries: `emgm`, `instantgenerics`, `multirec`, `regular`, and `syb`. We report on the latter four in this entry; `emgm` has its own entry (→ 5.8.3).

`instant-generics` Using type families and type classes in a way similar to `multirec` and `regular`, `instant-generics` is yet another approach to generic programming, supporting a large variety of datatypes and allowing the definition of type-indexed datatypes. It was first described by Chakravarty et al., and forms the basis of our new rewriting library. The current release of `instant-generics` on Hackage is a minimal version designed mostly to support our new rewriting package. This new rewriting library supports conditional guards on the rewrite rules and allows metavariables to range over any types, unlike our previous rewriting library.

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

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

Unfortunately, `multirec` does not work well with ghc-6.12, due to a change in the expansion of type families that will hopefully be reverted in the future. If you are using `multirec`, then for the moment, we advise to use ghc-6.10.

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

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

syb Scrap Your Boilerplate (syb) has been supported by GHC since the 6.0 release. This library is based on combinators and a few primitives for type-safe casting and processing constructor applications. It was originally developed by Ralf Lämmel and Simon Peyton Jones. Since then, many people have contributed with research relating to syb or its applications.

Since syb has been separated from the base package, it can now be updated independently of GHC. We have recently released version 0.2 on Hackage, which has reintegrated the testsuite and introduced new generic producers, along with smaller changes and fixes.

We also continue to look at benchmarking and improving the performance of different libraries for generic programming (→ 5.8.4). Recently we have also investigated how to integrate generics in the Utrecht Haskell Compiler (→ 2.4).

Further reading

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

5.8.3 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 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 the 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. The article from these lectures has been extended and published as a technical report.

Recent Development

No changes have been made since the previous report.

Future plans

We plan to continue developing EMGM and to explore the use of this library in many domains. We welcome ideas or contributions from the community.

Contact

Let us know 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 the package and browse the API at the Hackage page.

5.8.4 Optimizing generic functions

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

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

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

We have benchmarked four generic programming libraries: `emgm`, `syb`, `multirec`, and `regular`. We compare different generic functions in each of these libraries to a hand-written version. We have concluded that inlining plays a crucial role in the optimization of generics. Previously we used flags to increase the chances of the GHC inliner to optimize our functions. However, such flags change the behavior of the inliner for the entire set of modules being compiled, which might have detrimental effects on performance. Currently we are investigating how to localize these hints to the compiler by using `INLINE` pragmas, for the `instant-generics` and `regular` generic programming libraries in particular.

In most cases, we can achieve very good performance results by providing `INLINE` pragmas to the conversion functions (*from* and *to*) for each datatype and for each instance of the generic function on a representation type (such as *Sum*, *Prod*, etc.). We have to be careful with the optimization phases, as sometimes inlining too early can prevent later optimizations. In this way, we achieve the same performance as a type-specific hand-written version for functions like *show* and *update*, using only the infrastructure that GHC already provides. The performance of generic *read* is also significantly improved.

Unfortunately, some generic functions are still difficult to optimize with this technique. In particular, functions which involve additional datatypes in their type (such as *enum*, which returns a *list* of elements) prevent proper optimization. We are currently looking into how we can circumvent this restriction. We also plan to update our libraries to add the necessary pragmas for increased efficiency, but since we require the new inliner we have to wait until GHC version 6.14 is released.

Further reading

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

5.9 User interfaces

5.9.1 Gtk2Hs

Report by:	Axel Simon
Participants:	Andy Stewart and many others
Status:	beta, actively developed

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

GUIs written using Gtk2Hs use themes to resemble the native look on Windows. Gtk is the toolkit used by Gnome, one of the two major GUI toolkits on Linux. On Mac OS programs written using Gtk2Hs are run by Apple's X11 server but may also be linked against a native Aqua implementation of Gtk.

Gtk2Hs features:

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

While that last six months have seen the addition of several new functions and even whole libraries, one of the most visible changes is that Gtk2Hs is now available as a set of Cabal packages. This transition also means that many of the additional libraries that reside in the Gtk2Hs repository will move into their own repositories and can be maintained by people outside the Gtk2hs core team. Cabal packages also mean that no more Windows installer is needed and that building the code is more resilient to changes to the way GHC manages its build process: These issues are now all dealt with by Cabal, thereby greatly simplifying the installation and maintenance of Gtk2Hs. The separation into many Cabal files also makes it possible to use just the Cairo package to render vector graphics into PNGs or to use Cairo and Pango to produce Unicode PDF documents!

Another important addition is the support for correct garbage collection in multi-threaded programs. This has bitten several users who wanted to write

multi-threaded programs and became a more pressing issue with GHC's support for concurrent garbage collection.

Gtk2Hs version 0.11.0 has been released on May 25th. It has been tested on Linux, Mac OS X, and Windows (XP/7), and works in GHCi and multi-threaded programs.

Further reading

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

5.9.2 CmdArgs

Report by:	Neil Mitchell
Status:	released

CmdArgs is a library for defining and parsing command lines. The focus of CmdArgs is allowing the concise definition of fully-featured command line argument processors, in a mainly declarative manner (i.e., little coding needed). Compared to the standard GetOpt library, CmdArgs is often about three times shorter. CmdArgs also supports multiple mode programs, for example as used in git/darcs/Cabal.

Further reading

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

5.10 Graphics and Music

5.10.1 LambdaCube

Report by:	Csaba Hruska
Status:	experimental, active development

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

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

The most important features are the following:

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

If your system has OpenGL and GLUT installed, the `lambdacube-examples` package should work out of the box. The engine is also integrated with the Bullet physics engine (→ 6.13.3), and you can find a running example in the `lambdacube-bullet` package.

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

Further reading

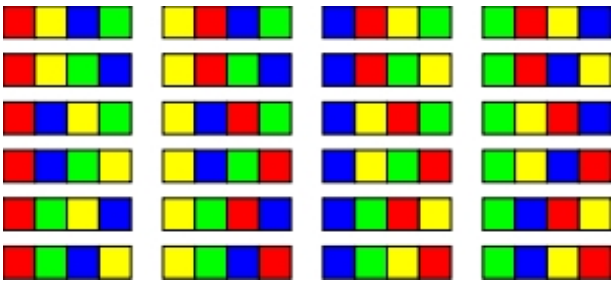
http://www.haskell.org/haskellwiki/LambdaCubeEngine

5.10.2 diagrams

Report by:	Brent Yorgey
Status:	active development

The diagrams library provides an embedded domain-specific language for creating simple pictures and diagrams. 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. The overall vision is for diagrams to become a viable alternative to DSLs like MetaPost or Asymptote, but with the advantages of being *purely functional* and *embedded*.

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/.

The library is currently undergoing a major rewrite, in order to use a more flexible constraint-solving layout engine and abstract out the rendering backend (the current version depends solely on the Cairo library for rendering). Other planned features include animation support, more sophisticated paths and path operations, and an xmonad-like core/contrib model for incorporating user-submitted extension modules.

Further reading

- http://code.haskell.org/diagrams/
- http://byorgey.wordpress.com/2009/09/24/diagrams-0-2-1-and-future-plans/
- http://www.tug.org/metapost.html
- http://asymptote.sourceforge.net/

5.10.3 GPipe

Report by:	Tobias Bexelius

GPipe models the entire graphics pipeline in a purely functional, immutable and type-safe way. It is built on top of the programmable pipeline (i.e., non-fixed function) of OpenGL 2.1 and uses features such as vertex buffer objects (VBO's), texture objects, and GLSL shader code synthetization to create fast graphics programs. Buffers, textures, and shaders are cached internally to ensure fast framerate, and GPipe is also capable of managing multiple windows and contexts. GPipe's aim is to be as close to the conceptual graphics pipeline as possible, and not to add any more levels of abstraction.

In GPipe, you work with four main data types: PrimitiveStreams, FragmentStreams, FrameBuffers, and textures. They are all immutable, and all parameterized on the type of data they contain to ensure type safety between pipeline stages. By creating your own instances of GPipes type classes, it is possible to use additional data types on the GPU.

Version 1.2.1 with documentation is released on Hackage, as well as some utility libraries that enable loading of Collada geometries and JPEG textures. There are also a few examples and tutorials that can be found through the wiki.

I am not currently working on any more additions myself, but the sources are available on github and anyone is welcome to contribute.

Further reading

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

5.10.4 ChalkBoard

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

ChalkBoard is a domain specific language for describing images. The language is uncompromisingly functional and encourages the use of modern functional idioms. The novel contribution of ChalkBoard is that it uses off-the-shelf graphics cards to speed up rendering of our functional description.

We always intended to use ChalkBoard to animate educational videos, as well as for processing streaming videos. Since the last HCAR report, we have used ChalkBoard in two main projects, covering both these goals.

- We used ChalkBoard to post-process a "special feature" presentation at PEPM'10, where we turned a video of KU actors (err, us) giving a presentation, into individual frames, and processed these frames using ChalkBoard to add clearer slides, and some animations.
- We are working on a new animation language, based round a new applicative functor, `Active`. It has been called Functional Reactive Programming, without the reactive part!

We talked about a case study of using our `Active` language at TFP in May, when Kevin gave the talk "Every Animation Should Have a Beginning, a Middle, and an End".

Further reading

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

5.10.5 graphviz

Report by:	Ivan Lazar Miljenovic
Status:	version 2999.9.0.0

The graphviz library provides Haskell bindings for the *Graphviz* suite of tools for visualizing graphs by utilizing Graphviz's *Dot* language. The major features of the graphviz library include:

- Almost complete coverage of all Graphviz attributes and syntax.
- Support for specifying clusters.
- The ability to use a custom node type.
- Functions for running a Graphviz layout tool with all specified output types.
- The ability to not only generate but also parse Dot code with two options: strict and liberal (in terms of ordering of statements).
- Functions to convert FGL graphs to Dot code — including support to group them into clusters — with a high degree of customization by specifying which attributes to use and limited support for the inverse operation.

- Round-trip support for passing an FGL graph through Graphviz to augment node and edge labels with positional information, etc.

For a sample graph visualized using the graphviz library, see SourceGraph (→ 4.2.2).

Further reading

- http://projects.haskell.org/graphviz/
- http://hackage.haskell.org/package/graphviz
- http://www.graphviz.org/

5.10.6 Euterpea

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

See: http://haskell.org/communities/05-2009/html/report.html#sect5.12.2.

5.11 Web and XML programming

5.11.1 Haskell XML Toolbox

Report by:	Uwe Schmidt
Status:	seventh major release (current release: 8.5.2)

Description

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

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

HXT is partitioned into 6 packages: The base package `hxt`, the package for the old approach working with filter `hxt-filter` (this one will not be further developed), the package `hxt-xpath` for XPath functionality, `hxt-xslt` for the XSLT interpreter, `hxt-binary` a small package for binary (de-)serialization of HXT DOM trees, and `hxt-cache`, a package for caching XML/HTML documents in parsed format.

Features

- Validating XML parser
- Very liberal HTML parser
- Lightweight lazy parser for XML/HTML based on Tagsoup (→ 5.11.3)
- 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
- Git repository with current development versions of all packages http://git.fh-wedel.de/repos/hxt.git

Current Work

Currently mainly maintenance work is done. This includes space and runtime optimizations.

The HXT library is extensively used in the Holumbus project (→ 6.3.1), there it forms the basis for the index generation. Development is currently driven by the needs of the Holumbus project.

Further reading

The Haskell XML Toolbox Web page (http://www.fh-wedel.de/~si/HXmlToolbox/index.html) includes downloads, online API documentation, a cookbook with nontrivial examples of XML processing using arrows and RDF documents, and master 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).

5.11.2 Hawk

Report by:	Uwe Schmidt
Participants:	Björn Peemöller, Stefan Roggensack, Alexander Treptow
Status:	first release

The Hawk system is a web framework for Haskell. It is comparable in functionality and architecture with Ruby on Rail and other web frameworks. Its architecture follows the MVC pattern. It consists of a simple relational database mapper for persistent storage of data and a template system for the view component. This template system has two interesting fea-

tures: First, the templates are valid XHTML documents. The parts where data has to be filled in are marked with Hawk specific elements and attributes. These parts are in a different namespace, so they do not destroy the XHTML structure. The second interesting feature is that the templates contain type descriptions for the values to be filled in. This type information enables a static type check whether the models and views fit together.

A first application of the Hawk framework is a customizable search for Hayoo! (→ 6.3.1). But the framework is independent of the Holumbus search engine. It will be applicable for the development of arbitrary web applications.

Hawk was developed by Björn Peemöller and Stefan Roggensack. Currently, Alexander Treptow is applying, testing, and extending the framework.

5.11.3 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 Hoogle (http://haskell.org/communities/05-2009/html/report.html#sect4.4.1).

A new version of tagsoup has been released, fully supporting the HTML 5 specification. The API also has experimental support for ByteString (although currently ByteString is slower than String).

Further reading

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

5.11.4 BlazeHtml

Report by:	Jasper Van der Jeugt
Participants:	Simon Meier, Chris Done, Fred Ross, Jim Whitehead, Harald Holtmann, Oliver Mueller and Tom Harper
Status:	in development

BlazeHtml is a blazingly fast HTML combinator library.

Our main goal is to push Haskell as a web development language. Compared to the popular languages currently used for web development (php, Ruby, Python) Haskell has the advantages of speed and type-safety.

To write a web application, at least three components are required: A web application server, a data storage layer, and an HTML generation library. This library addresses the last of these three components.

We want to provide a set of combinators with which the user can describe HTML documents in an abstract way. Our main focus is on efficiency, and initial benchmarks have already pointed out that we can be a lot faster than heavily optimized libraries in other programming languages. Furthermore, we want to guarantee correctness (validity) of the produced documents. Composability is also a key feature, since all documents are first-class Haskell values.

This library originated on ZuriHac 2010, and has been under heavy development since. It was accepted as a project for Google Summer of Code 2010, meaning that the community can expect it to be stable, usable, and well-documented in August.

Currently, we are trying to get a very strong performance baseline. After that, we can add more abstraction and features to the library. Then, the library will be completed with a set of benchmarks, tests, and tutorials.

The project is, in its current state, accessible through the GitHub repository. However, it is not stable enough for real-world use (yet).

Further reading

http://github.com/jaspervdj/BlazeHtml

5.11.5 WAI

Report by:	Michael Snoyman
Status:	experimental

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

In designing this package, performance was first priority: there should be no performance overhead for using the WAI. As such, an enumerator interface was selected for the response body, a handle-like interface, called a source, for the request body, and bytestrings used throughout.

In addition, to promote type safety, datatypes such as RequestHeader or Status are used instead of raw ByteStrings and Ints. Finally, this interface has been kept as general as possible by excluding variables which are not universal to all web servers.

WAI is not set in stone; work has begun on the next version. However, do not let this prevent you from using WAI right now: the upcoming changes will be minor, and the Package Versioning Policy is being followed strictly, so old code will not be broken. All input is taken very seriously, so send in your suggestions.

Hopefully, WAI can be one of many smaller packages which lead to collaboration in the Haskell web development community and development of a healthy ecosystem. (See also Yesod (→ 6.3.6).)

Further reading

http://github.com/snoyberg/wai

6 Applications and Projects

6.1 For the Masses

6.1.1 Darcs

Report by:	Eric Kow
Status:	active development

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

Our most recent major release, Darcs 2.4, was in January 2010. It provides faster repository-local operations, a new interactive hunk editing feature among other bug fixes and features. For our next release, we hope to continue the trend of improving Darcs performance:

1. Better support for long histories: Petr Rockai has begun work (originally started by David Roundy) to make Darcs handle long histories in hashed repositories. If you tag your repositories regularly, operations that add or remove patches to Darcs should take O(1) time instead of O(N) with respect to the number of the patches in your history.

2. Faster Darcs annotate: Benedikt Schmidt has nearly completed his work on a new "patch index" feature which we hope to make darcs annotate considerably faster. He also plans to overhaul the user interface to provide more human-readable output.

These changes and more will appear in the upcoming Darcs 2.5 release, scheduled for July 2010. Also, we are excited to report that a Darcs project has been accepted for the 2010 Google Summer of Code. Alexey Levan will be working to improve Darcs performance over networks. Meanwhile, we still have a lot of progress to make and are always open to contributions. Haskell hackers, we need your help!

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

Further reading

http://darcs.net

6.1.2 xmonad

Report by:	Gwern Branwen
Status:	active development

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

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

Details of changes between releases can be found in the release notes:

- http://haskell.org/haskellwiki/Xmonad/Notable_changes_since_0.7
- http://haskell.org/haskellwiki/Xmonad/Notable_changes_since_0.8
- http://haskell.org/haskellwiki/Xmonad/Notable_changes_since_0.9
- XMonad.Config.PlainConfig allows writing configs in a more 'normal' style, and not raw Haskell
- Supports using local modules in xmonad.hs; for example: to use definitions from ~/.xmonad/lib/XMonad/Stack/MyAdditions.hs
- xmonad –restart CLI option
- xmonad –replace CLI option
- XMonad.Prompt now has customizable keymaps
- Actions.GridSelect - a GUI menu for selecting windows or workspaces
- Actions.OnScreen
- Extensions now can have state
- Actions.SpawnOn - uses state to spawn applications on the workspace the user was originally on, and not where the user happens to be
- Markdown manpages and not man/troff
- XMonad.Layout.ImageButtonDecoration & XMonad.Util.Image
- XMonad.Layout.Groups

- XMonad.Layout.ZoomRow
- XMonad.Layout.Renamed
- XMonad.Layout.Drawer
- XMonad.Hooks.ScreenCorners
- XMonad.Actions.DynamicWorkspaceOrder
- XMonad.Actions.WorkspaceNames
- XMonad.Actions.DynamicWorkspaceGroups

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

Further reading

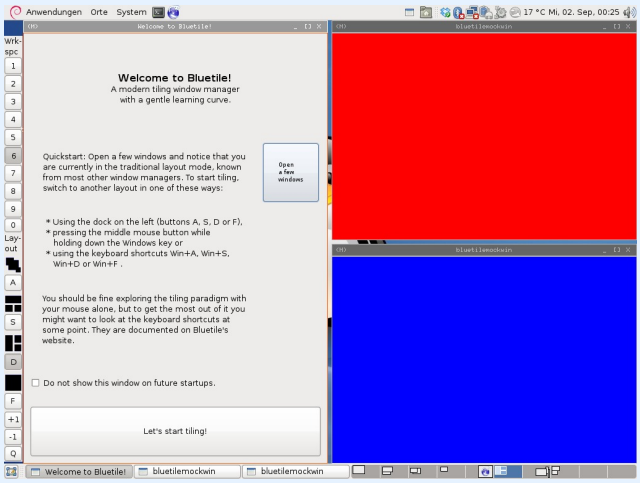
- Homepage: http://xmonad.org/
- Darcs source:
  `darcs get` http://code.haskell.org/xmonad
- IRC channel: `#xmonad @ irc.freenode.org`
- Mailing list: ⟨xmonad@haskell.org⟩

6.1.3 Bluetile

Report by:	Jan Vornberger
Status:	active development

Bluetile is a tiling window manager for X based on xmonad ((→ 6.1.2)). Windows are arranged to use the entire screen without overlapping. Bluetile's focus lies on making the tiling paradigm easily accessible to users coming from traditional window managers by drawing on known conventions and providing both mouse and keyboard access for all features. It also tries to be usable "out of the box", requiring minimal to no configuration in most cases.

- Hybrid approach: Stacking window layout & tiling layouts available
- Maximizing & minimizing windows in all layouts
- All features accessible from mouse, as well as keyboard
- Good multihead support
- Proper handling of fullscreen applications
- Designed to integrate with the GNOME desktop environment



Further reading

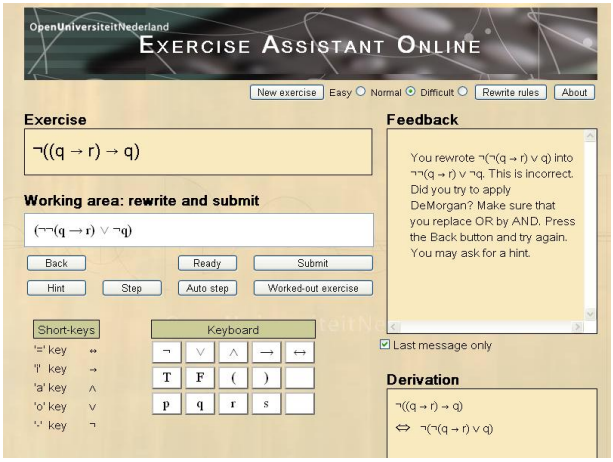
http://www.bluetile.org/

6.2 Education

6.2.1 Exercise Assistants

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

At the Open Universiteit Nederland and Universiteit Utrecht we are continuing our work on tools that support students in solving exercises incrementally by checking intermediate steps. 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 equations, reducing matrices to echelon normal form, and simplifying expressions in relation algebra (among others).



We have been working on exercise assistants for learning how to program in Haskell. A case study was performed to use programming strategies for automatically assessing student programs submitted for a first-year course on functional programming in Utrecht. This is ongoing research.

We have further integrated our tools with the Digital Mathematics Environment (DWO) of the Freudenthal Institute and the ActiveMath learning system (DFKI and Saarland University). Both environments offer a rich collection of interactive exercises for practicing exercises in mathematics. We have extended these exercises with our facility to automatically generate hints and worked-out examples. In the last couple of months, support for solving inequalities and rewriting expressions involving powers has been added to our tools.

We have recently updated the Cabal source package of our feedback services.

Further reading

- Online exercise assistant, accessible from our project page.
- Bastiaan Heeren, Johan Jeuring and Alex Gerdes. Specifying Rewrite Strategies for Interactive Exercises. Mathematics in Computer Science, 3(3):349–370, 2010.
- Alex Gerdes, Johan Jeuring, and Bastiaan Heeren. Using Strategies for Assessment of Programming Exercises. Technical Symposium on Computer Science Education (SIGCSE 2010).

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 Yahc

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

The first course on algorithms in CS at *Universidad Nacional de Córdoba* is centered on the derivations of algorithms from specifications, as proposed by R.S. Bird (*Introduction to functional programming using Haskell*, Prentice Hall Series in Computer Science, 1998), E.W. Dijkstra (*A Discipline of Programming*, Prentice Hall, 1976), and R.R. Hoogerwoord (*The design of functional programs: a calculational approach*, Technische Universiteit Eindhoven, 1989). To achieve this goal, students should acquire the ability to manipulate complex predicate formulae; thus the students first learn how to prove theorems in a propositional calculus similar to the equational propositional logic of D. Gries and F.B. Schneier (*A Logical Approach to Discrete Math*, Springer-Verlag, 1993).

During the semester students make many derivations as exercises and it is helpful for them to have a tool for checking the correctness of their solutions. Yahc checks the correctness of a sequence of applications of some axioms and theorems to the formulae students are trying to prove. The student starts a derivation by entering an initial formula and a goal and then proceeds by telling Yahc which axiom will be used and the expected outcome of applying the axiom as a rewrite rule; if that rewriting step is correct then the process continues until the student reaches the goal.

After the experience gained during one semester we made some changes in the user-interface. We have also added the definition of new constants and rules, which permits the resolution of logical puzzles.

In the long term we plan to consider an equational calculus with functions defined by induction over lists and natural numbers.

Further reading

http://www.cs.famaf.unc.edu.ar/~mpagano/yahc/

6.2.4 grolprep

Report by:	Dino Morelli
Participants:	Betty Diegel
Status:	experimental, actively developed

grolprep is a web application for studying the FCC GROL questions in preparation of taking the exams.

The study of this multiple-choice data is in the flash-card style. Students can choose from Elements 1, 3 and 8 and can specify any subelement of those three for specific study. Questions and answers can be randomly presented.

Additionally, simulations of the randomly-chosen exams can be practiced with this software.

grolprep will shortly be used by students of Avionics program at the Burlington Aviation Technology School.

Further reading

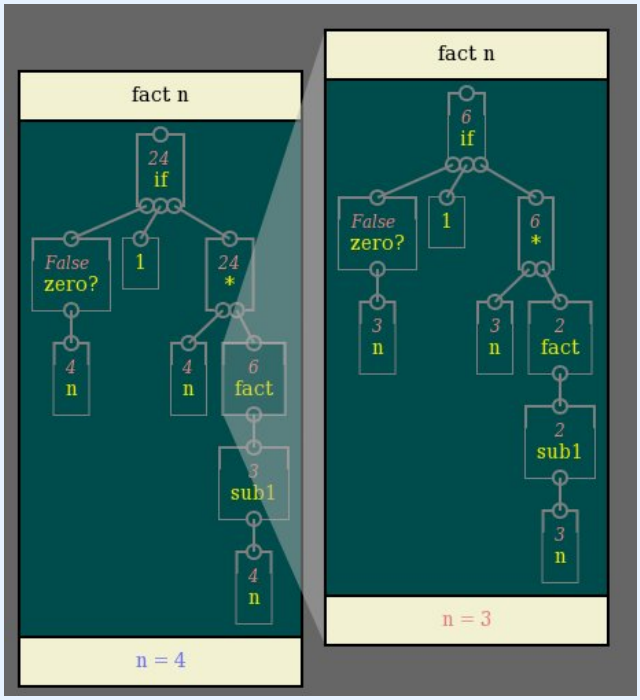
- Live website: http://ui3.info/grolprep/bin/fcc-grol-prep.cgi
- Project page: http://ui3.info/d/proj/grolprep.html
- Source repository: `darcs get` http://ui3.info/darcs/grolprep

6.2.5 Sifflet

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

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

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



Features

- Visual editor.
- Visual tracer/debugger which shows how function calls are evaluated. This supports an active learning process: Sifflet does not overwhelm students with a huge trace of function calls; it provides only as much expansion as the student requests.
- Extensive tutorial with 6,348 words and 31 pictures.
- Number, string, and list data types.
- A function "palette" with a small number of primitive functions.
- Runnable examples of compound functions.

Availability

Sifflet made its public debut in May, 2010. It is available from Hackage: http://hackage.haskell.org/package/sifflet

Future plans

The next release will provide a few usability improvements. Longer-term plans include:

- Export code to Haskell and other programming languages.
- Type inference.
- Higher-order functions.
- Tree data and/or user-defined data types.

Further reading

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

6.3 Web Development

6.3.1 Holumbus Search Engine Framework

Report by:	Uwe Schmidt
Participants:	Timo B. Hübel, Sebastian Gauck, Stefan Schmidt, Björn Peemöller, Stefan Roggensack, Sebastian Reese, Alexander Treptow
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 (→ 5.11.1).

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
- Git repository containing the current development version of all packages under http://holumbus.fh-wedel.de/src.git
- Distributed building of search indexes

Current Work

The data structures of the Holumbus indexes have been optimized for space and time. There is a new and efficient prefix tree structure, which further enables index updates.

The indexer and search module is used to support the Hayoo! engine for searching the hackage package library (http://holumbus.fh-wedel.de/hayoo/hayoo.html). Because of the fast growing number of packages on hackage, the Hayoo! search engine will be extended by a package search.

Sebastian Reese has finished his work on applying the MapReduce framework and for giving tuning and configuration hints. Benchmarks for various small problems and for generating search indexes have shown that the architecture scales very well.

In a subproject of Holumbus, the so called Hawk framework (→ 5.11.2), Björn Peemöller and Stefan Roggensack have developed a web framework for Haskell. Currently Alexander Treptow is applying, testing, and extending the framework. A first application is a customizable search for Hayoo!

Further reading

The Holumbus web page (http://holumbus.fh-wedel.de/) includes downloads, Git web interface, current status, requirements, and documentation. Timo Hübel's master thesis describing the Holumbus index structure and the search engine is available at http://holumbus.fh-wedel.de/branches/develop/doc/thesis-searching.pdf. Sebastian Gauck's thesis dealing with the crawler component is available at http://holumbus.fh-wedel.de/src/doc/thesis-indexing.pdf The thesis of Stefan Schmidt describing the Holumbus MapReduce is available via http://holumbus.fh-wedel.de/src/doc/thesis-mapreduce.pdf.

6.3.2 HCluster

Report by:	Alberto Gómez Corona

HCluster (provisional name) is a remote clustering middleware aimed initially at verifying online and offline computations in distributed electoral processes. Extended to permit clustering with distributed transactions and cloud computing.

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

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 with periodic synchronization with certain restrictions. The restrictions depend on algebraic properties of the transactions.

Distribution of data and distributed transactions are possible. 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. Testing synchronization and online clustering now defined and coded the model for distributed transactions.

Future plans

- test distributed transactions
- create internet documentation

Contact

⟨agocorona@gmail.com⟩

6.3.3 gitit

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

Gitit is a wiki built on Happstack and backed by a git, darcs, or mercurial filestore. Pages and uploaded files can be modified either directly via the VCS's command-line tools or through the wiki's web interface. Pandoc (→ 6.4.1) is used for markup processing, so pages may be written in (extended) markdown,

reStructuredText, LaTeX, HTML, or literate Haskell, and exported in eleven different formats, including LaTeX, ConTeXt, DocBook, RTF, OpenOffice ODT, MediaWiki markup, and PDF.

Notable features of gitit include:

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

Further reading

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

6.3.4 Happstack

Report by:	Jeremy Shaw

Happstack, the Haskell Application Server Stack, is loosely defined as a web development framework. It includes a web server, multiple systems for routing incoming requests to handlers, integration with a variety of templating systems, a persistent data storage layer, and more!

Instead of having to configure Apache, MySQL, PHP, etc., you just deploy your self-contained binary and run it! We do also support integration with MySQL and Apache if that is what your app demands.

The happstack persistent storage layer (often referred to as MACID) gives you the power of ACID databases, with the convenience of Haskell data types and functions. You can use normal Haskell data types, and your queries are written in pure Haskell. Work with trees, maps, graphs, and your other favorite types with ease!

Future plans

The big focus now is on improving the Happstack documentation. We also plan to continue work on MACID by improving replication support, and adding support for sharding.

Further reading

http://www.happstack.com/

6.3.5 Mighttpd — yet another Web server

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

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

c10k Since GHC is using the select system call, a Haskell program complied with GHC cannot handle over 1,024 connections/files simultaneously. The c10k package uses the prefork technique to get rid of this barrier.

webserver The webserver package provides HTTP parser, session management, redirection, CGI, and so on. This package is independent from back-end storage systems. So you can build a Web server on any storage system including files, key-value-store DB, etc.

mighttpd This package provides a simple but practical web server based on files using the c10k and webserver packages.

I am planning to implement FastCGI and WebSocket.

Further reading

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

6.3.6 Yesod

Report by:	Michael Snoyman
Status:	experimental

Yesod is a web framework designed to play towards the strengths of the Haskell language to make web programming safer and more productive. It is fair to say that most web development today occurs in dynamic languages like PHP, Python, and Ruby, and we see the results: cross-site scripting attacks, applications that do not scale, and countless minor bugs entering production because they can only be detected at runtime.

Yesod itself, however, provides very little functionality. Instead of bundling features into the main package, useful features have been spun off so that they are usable outside of Yesod whenever possible. Packages for authentication, client-side encrypted session data, middlewares, web encodings, YAML, and more are all fully available on Hackage, without any reliance on Yesod.

The second major version of Yesod is currently being written. In collaboration with others in the community, this release will see even more features factored out: the controller has become web-routes-quasi. This split benefits Yesod as well: it will be gaining type-safe URLs and pluggable components. A new templating system, Hamlet, which is fully type-checked and prop-

erly handles the aforementioned type-safe URLs, has also been released.

Now is a great time to get involved in the project. There is a brand new site (http://docs.yesodweb.com/) which will provide in-depth documentation on Yesod and many of its related packages. Check out what is there, e-mail in suggestions and features you would like to see, and send in your patches!

Further reading

http://docs.yesodweb.com/

6.3.7 Lemmachine

Report by:	Larry Diehl
Participants:	Jason Dusek
Status:	experimental, active development

Lemmachine is a REST'ful web framework that makes it easy to get HTTP right by exposing users to overridable hooks with sane defaults. The main architecture is a copy of Erlang-based Webmachine, which is currently the best documentation reference (for hooks & general design).

Lemmachine stands out from the dynamically typed Webmachine by being written in dependently typed Agda (→ 3.2.2). The goal of the project is to show the advantages gained from compositional testing by taking advantage of proofs being inherently compositional. See http://github.com/larrytheliquid/Lemmachine/blob/master/src/Lemmachine/Default/Proofs.agda for examples of universally quantified proofs (tests over all possible input values) written against the default resource, which does not override any hooks.

When a user implements their own resource, they can write simple lemmas ("unit tests") against the resource's hooks, but then literally reuse those lemmas to write more complex proofs ("integration tests"). For examples see some reuse of lemmas in the proofs.

The big goal is to show that in service oriented architectures, proofs of individual middlewares can themselves be reused to write cross-service proofs (even higher level "integration tests") for a consumer application that mounts those middlewares. See a post at http://vision-media.ca/resources/ruby/ruby-rack-middleware-tutorial for what is meant by middleware.

Another goal is for Lemmachine to come with proofs against the default resource (as it already does). Any hooks the user does not override can be given to the user for free by the framework! Anything that is overridden can generate proofs parameterized only by the extra information the user would need to provide. This would be a major boost in productivity compared to traditional languages whose libraries cannot come with tests for the user that have language-level semantics for real proposition reuse!

Lemmachine currently uses the Haskell Hack abstraction so it can run on several Haskell webservers. Because Agda compiles to Haskell and has an FFI, existing Haskell code can be integrated quite easily.

The project is still in development and rapidly changing. Lemmas and proofs exist for status resolution, and you can now run resources! The focus will now comprise of a gradual direct translation of RFC 2616 sections into dependent type theory.

Further reading

http://github.com/larrytheliquid/Lemmachine

6.3.8 Snap

Report by:	Doug Beardsley
Participants:	Gregory Collins, Shu-yu Guo, James Sanders

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

The project's initial release consisted of a low-level web server API, a fast web server that includes an optional high-concurrency libev backend, and an XML-based tempting system. We also placed special emphasis on clean code, good test coverage, and quality documentation and tutorials.

Future plans

The next step for Snap is the development of a component system that allows web apps to be constructed from modular pieces. This will lay the groundwork for higher-level functionality such as session management, form handling, administration console, data persistence, etc. The component system will allow the abstraction of functionality and enable different concrete implementations to be used interchangeably.

Further reading

http://snapframework.com

6.4 Data Management and Visualization

6.4.1 Pandoc

Report by:	John MacFarlane
Participants:	Andrea Rossato, Peter Wang, Paulo Tanimoto, Eric Kow, Luke Plant, Justin Bogner
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.

There have been several releases since the last report, with many bug fixes and small improvements. There are two big architectural changes. First, pandoc no longer requires Template Haskell, which should make it more portable. Second, a new, flexible template system has been added, allowing users much more control over document headers and footers. Other major changes include support for xetex, support for reST tables, support for tables without header rows, support for formatting math as MathML, a new "plain text" output format, and a much more permissive HTML parser. The old `hsmarkdown` and `html2markdown` scripts have been removed; `pandoc` itself can now do the work of `html2markdown`. Summaries of the new features in each release are available on the (newly redesigned) website, along with full documentation and a new tutorial on using the pandoc library for structured text manipulation.

Further reading

http://johnmacfarlane.net/pandoc/

6.4.2 HaExcel — From Spreadsheets to Relational Databases and Back

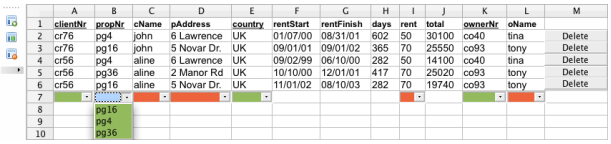
Report by:	Jácome Cunha
Participants:	João Saraiva, Joost Visser
Status:	unstable, work in progress

HaExcel is a framework to manipulate and transform 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 and refactor spreadsheets.

Using part of HaExcel, we developed an OpenOffice Calc (http://www.openoffice.org/product/calc.html) addon. Its back-end reuses part of HaExcel and its front-end is written in OpenOffice Basic. This addon allows the integration of a relational model into the spreadsheet. Using this model the user gets three new features in the spreadsheet environment: *auto-completion* of columns, that is, choosing values of some columns, other columns become automatically completed; *safe deletion* of rows where the user is warned when deleting important information; and *no edition* of columns that could compromise the data integrity. All the features can be enabled and disabled by the user at any time. A snapshot of a spreadsheet with the addon can be seen below.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	clientNr	propNr	cName	pAddress	country	rentStart	rentFinish	days	rent	total	ownerNr	oName	
2	cr76	pg4	john	6 Lawrence	UK	01/07/00	08/31/01	602	50	30100	co40	tina	Delete
3	cr76	pg16	john	5 Novar Dr.	UK	09/01/01	09/01/02	365	70	25550	co93	tony	Delete
4	cr56	pg4	aline	6 Lawrence	UK	09/02/99	06/10/00	282	50	14100	co40	tina	Delete
5	cr56	pg36	aline	2 Manor Rd	UK	10/10/00	12/01/01	417	70	25020	co93	tony	Delete
6	cr56	pg16	aline	5 Novar Dr.	UK	11/01/02	08/10/03	282	70	19740	co93	tony	Delete
7													
8		pg16											
9		pg4											
10		pg36											

More about this can be read in the paper "Discovery-based Edit Assistance for Spreadsheets".

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

We are currently exploring foreign key constraints from their detection to their migration to the generated spreadsheet.

Further reading

http://www.di.uminho.pt/~jacome

6.4.3 Ferry (Database-Supported Program Execution)

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

With project *Ferry* we try to establish a connection between two somewhat distant shores: programming languages and database technology. Ferry explores how far we can push the idea of relational database engines that directly and seamlessly participate in program evaluation to support the super-fast execution of data-intensive programs written in a variety of (functional) programming languages. Relational database systems (RDBMSs) provide the best understood and most carefully engineered query processing infrastructure available today. Notwithstanding these data processing capabilities, RDBMSs are often operated as plain stores that do little more than reproduce stored data items for further processing outside the database

host. With Ferry, instead, we aim to turn the database system into an efficient, capable, and highly scalable co-processor for your programming language's runtime. To this end, we search for, design, and implement new compilation strategies that map data types (e.g., nested and ordered lists, arrays, dictionaries), control structures (e.g., nested iteration, conditionals, variable assignment and reference), and idioms prevalent in functional programming and scripting languages into efficient database queries. Here, we try to push the limits of what has been considered possible (this includes algebraic data types, pattern matching, higher-order functions, and closures, to name a few).

Variants of the Ferry technology have been used

- to enhance the SQL code generator in Philip Wadler's *Links*, such that a significantly larger class of Links programs may be considered *databaseable* now, and
- to create a capable and efficient version of *LINQ to SQL* provider (plugging into the Microsoft .NET Language Integrated Query framework).

We are currently re-implementing the Ferry compiler in Haskell (using GHC). It will be published as an open source project soon.

Future plans

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

Further reading

http://www.ferry-lang.org

6.4.4 Sirenial

Report by: Martijn van Steenbergen

Sirenial is an embedded DSL for modelling SQL statements.

```
select t0.townName
from towns t0
where (t0.id = 1)
```

The above query is the result of executing *getTownName* 1, where *getTownName* is defined as follows:

```
getTownName :: Ref Db.Town → Query String
getTownName townId = do
  [townName] ← select $ do
    t ← from Db.tableTown
    restrict (t # Db.townId .==. expr townId)
    return (t # Db.townName)
  returntownName
```

The inner **do**-block is code in the *Select* monad, containing functions such as *from* and *restrict*, responsible for the creation of a single SELECT statement. These *Select* computations are lifted into the *Query* monad using *select*, where it becomes apparent that a query always yields a list of results.

The symbols prefixed with *Db* model the database schema:

```
data Town
tableTown  = Table "towns"
           :: Table Town
townName   = Field tableTown "townName"
           :: Field Town     String
townId     = Field tableTown "id"
           :: Field Town     (Ref Town)
```

An unusual feature of Sirenial is the automatic combining of queries: if similar *Query* computations are composed in applicative fashion, Sirenial will merge them transparently and send only a single statement to the database server. For example, executing *for* [11..20] *getTownName* results in only one SELECT statement being sent:

```
select t0.id, t0.townName
from towns t0
where t0.id in (11,12,13,14,15,16,17,18,19,20)
```

By locally replaying the WHERE clauses, the results from the database are distributed over the original *select* calls. All this happens transparently: the user can write the queries as if they were executed one by one.

The library is designed to be *moderately type-safe*: catch many mistakes at compile-time, yet use simple types, leading to simple and understandable type errors. It is easy to predict the generated SQL, as there is very little rewriting done on the statements. Finally, Sirenial is designed in such a way that it is easy to switch to using the library completely or partially, on existing databases and existing data.

Further reading

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

6.4.5 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. The system is being maintained by Oblomov Systems (→ 7.7).

- 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 edit operations on the WYSIWYG presentation of the document.
- Proxima is aware of the structure of the document. While editing the presentation of the document, the edit operations may also be structural. For example, a section can be changed into a subsection.

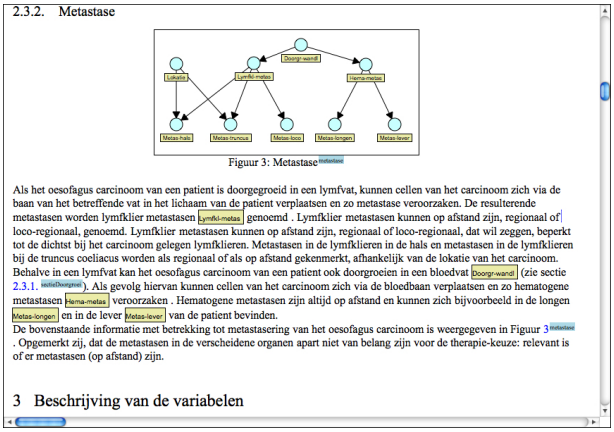
Another feature of Proxima is that it offers generic support for specifying content-dependent computations. For example, it is possible to create a table of contents of a document that is automatically updated as chapters or sections are added or modified.

Proxima 2.0

Proxima 2.0 provides a web-interface for Proxima. Instead of an application that renders onto a 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 an incremental rendering update back to the client. As a result, advanced editors can be created, which run in any browser. Among the current features of the system are drag and drop editing, session handling, and complex graphical presentations that may contain computed values and structures.

Because the (possibly large) HTML rendering may need to be communicated to the client on each key stroke or mouse gesture, Proxima 2.0 employs a number of techniques to ensure the editors respond fast enough over a network connection. On the one hand, *low bandwidth* may cause delays when sending large HTML renderings to the client. This problem is handled by using *incremental algorithms* to only send those parts of the rendering that were changed. On the other hand, *network latency* may cause a delay between a user edit gesture and the update received from the server. This problem is handled by using *predictive rendering*, which means that the client shows the predicted effect of the edit operation, until the actual update from the server is received and applied. Though both techniques may fail for pathological cases, they work very well for the majority of editors. As a result, the editors feel responsive enough even over remote network connections.

The Proxima website contains a gallery of live demo editors, as well as download instructions and documentation. The screenshot shows an editor for documenting Bayesian networks, running in Firefox.



Future plans

Proxima 2.0 is an open source project. We are looking for people who would like to participate.

Further reading

- http://www.cs.uu.nl/wiki/bin/view/Proxima
- http://www.oblomov.com

6.4.6 iTasks

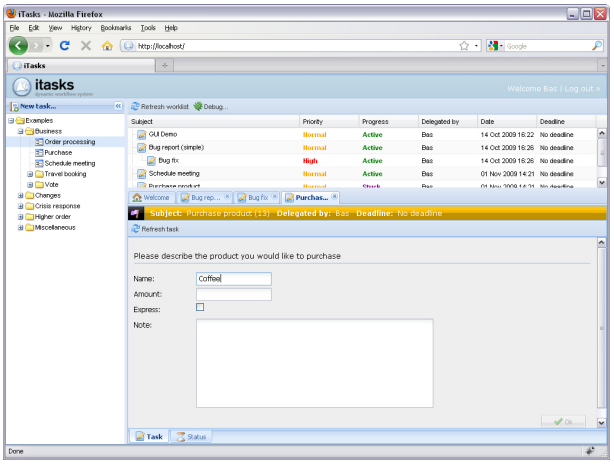
Report by:	Bas Lijnse
Participants:	Rinus Plasmeijer, Peter Achten, Pieter Koopman, Thomas van Noort, Jan Martin Jansen, Erik Crombag
Status:	active development

The iTask system provides a set of combinators to specify workflow in the pure and functional language Clean (→ 3.2.4) 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. Workflow specifications are supplemented with a generic foundation to generate executable multi-user workflow support systems that consist of a webservice-based server and a user-friendly Ajax client.

Compared to contemporary workflow systems, that often use simple graphical specification languages, the iTask system offers several advantages:

- Tasks are statically typed and can be higher-order.
- Combinators are fully compositional.

- Dynamic and recursive workflow is supported.
- Workflow instances can be modified during execution.



The iTask system makes extensive use of Clean's generic programming facilities for generating dynamic user-interfaces and data encoding/decoding.

Future plans

Currently, we are working on extending and stabilizing the iTask base system to a level where it can be used for serious applications. Additionally, we are working out case studies to explore use of iTasks to support real-world processes. We are also exploring how even more dynamic and unpredictable workflows can be supported.

Further reading

- http://itasks.cs.ru.nl/
- http://www.st.cs.ru.nl/Onderzoek/Publicaties/publicaties.html

6.5 Functional Reactive Programming

6.5.1 Functional Hybrid Modelling

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

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

Causal modelling languages are closely related to synchronous data-flow languages. They model system behavior using ordinary differential equations (ODEs) in explicit form. That is, cause-effect relationship between variables 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 modelling in a non-causal language more declarative (the focus is on expressing the equations in a natural way, not on how to express them to enable simulation) and also makes the models much more reusable.

FHM is an approach to modelling which combines functional programming and non-causal modelling. In particular, the FHM approach proposes modelling with first class models (defined by continuous DAEs) using combinators for their composition and discrete switching. The discrete switching combinators enable modelling of hybrid systems (i.e. systems that exhibit both continuous and discrete dynamic 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 and enables modelers to use the domain specific syntax in their models. The present prototype implementation of Hydra enables modelling with first class models and supports combinators for their composition and discrete switching.

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

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

Further reading

The implementation of Hydra is available from http://www.cs.nott.ac.uk/~ggg/ under the open source BSD license.

6.5.2 Elerea

Report by:	Patai Gergely
Status:	experimental, active

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 various higher-order constructs that also allow the user to work with arbitrary time-varying structures containing live signals.

Stateful signals can be safely generated at any time through a specialized monad, while stateless combinators can be used in a purely applicative style. Elerea signals can be defined recursively, and external input is trivial to attach. A unique feature of the library is that cyclic dependencies are detected on the fly and resolved by inserting delays dynamically, unless the user does it explicitly.

As an example, the following code snippet is a possible way to define an approximation of our beloved trig functions:

```
(sine,cosine) <- mdo
  s <- integral 0 c
  c <- integral 1 (-s)
  return (s,c)
```

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, because it is pull-based.

Version 1.1.0 introduced an experimental branch, which is a different implementation of the same basic idea. Unlike the current branch (which is to be deprecated in the near future), the experimental version preserves referential transparency. There are three variants to pick from in increasing order of complexity:

- Simple: discrete streams, which are isomorphic to functions over natural numbers (including the behavior of their class instances);
- Param: streams where a globally accessible input can be supplied to every node of the data-flow network in each sampling step; if this is a time step, we can simulate continuous-time streams to a certain extent — the catch being the lack of clear semantics;
- Delayed: the parametric variant augmented with the automatic delay feature, which allows users to be less explicit about how to make feedback loops well-formed (non-instantaneous). Note that referential transparency is lost again with this feature, which might or might not be acceptable depending on the application.

The code is readily available via cabal-install in the `elerea` package. You are advised to install `elerea-examples` as well to get an idea how to build non-trivial systems with it. The examples are separated in order to minimize the dependencies of the core library. The experimental branch is showcased by Dungeons of Wor, found in the `dow` package (→ 6.11.2). Additionally, the basic idea behind the experimental branch is laid out in the WFLP 2010 article *Efficient and Compositional Higher-Order Streams*.

Further reading

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

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 (http://haskell.org/communities/05-2009/html/report.html#sect5.6.2). The highlights are:

- a basic signal synthesis backend for Haskore (http://haskell.org/communities/05-2009/html/report.html#sect5.12.1),
- 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 checked several low-level implementations in order to achieve reasonable speed. We complement the standard list type with a lazy `StorableVector` structure and a `StateT s Maybe a` generator, like in stream-fusion. Now, both our custom signal generator type and the Stream type from stream-fusion can be fused to work directly on storable vectors.
- support for causal processes. Causal signal processes only depend on current and past data and thus are

suitable for real-time processing (in contrast to a function like time reversal). These processes are modeled as `mapAccumL` like functions. Many important operations like function composition maintain the causality property. They are important for sharing on a per sample basis and in feedback loops where they statically warrant that no future data is accessed.

Recent advances are:

- Lazy time values to be used for gate signals,
- enhanced type class framework for unifying lazy time values and signals expressed by lists, storable vectors or signal generators.
- Connection to alsa bindings, in order to provide real-time sound synthesis controlled by MIDI events from keyboards or sequencers,
- Stand-alone binding to Sox for audio format conversion and playback,
- A pyramid filter for efficient computation of moving average and moving maximum for baseline detection of mass spectra,
- A set of signal processors that generates maximally efficient vectorized code using LLVM as portable assembler.

Further reading

- http://www.haskell.org/haskellwiki/Synthesizer
- http://arxiv.org/abs/1004.4796

6.6.2 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 (→ 5.3.1), etc.). Once appropriate geometric primitives have been extracted by the image processing wrappers we can define interesting computations using elegant functional constructions. Recent work includes cabalization of the main modules.

Further reading

http://code.haskell.org/easyVision

6.6.3 n-Dimensional Volume Calculation for Non-Convex Polytops

Report by:	Farid Karimipour
Participants:	Mahmoud R. Delavar, Andrew U. Frank
Status:	active development

This is the continuation of the work "n-dimensional convex decomposition of polytops" (http://haskell.org/communities/11-2009/html/report.html#sect6.6.4) where we showed how to decompose an n-dimensional non-convex polytop to a set of convex components. The algorithm builds a tree of signed convex components that are stored as a set of n-simplexes: even levels are additive, whereas components in odd levels are subtractive. Here, the elements of this tree are utilized to calculate the volume of the original n-dimensional non-convex polytop ("volume" is used as a generalized term for all dimensions, i.e., "area" for 2D, etc.). The resultant components are triangulated whose volume calculation is straightforward:

$$V<(e_{01},\ \ldots,\ e_{0n}),\ \ldots,\ (e_{n1},\ \ldots,\ e_{m})> \;=\; \frac{1}{2}\begin{vmatrix} 1 & \ldots & 1 \\ e_{01} & \ldots & e_{n1} \\ \ldots & \ldots & \ldots \\ e_{0n} & \ldots & e_{m} \end{vmatrix}$$

Summing up the volumes of all triangles (tetrahedrons in 3D) will provide us with the volume of the n-dimensional non-convex polytop:

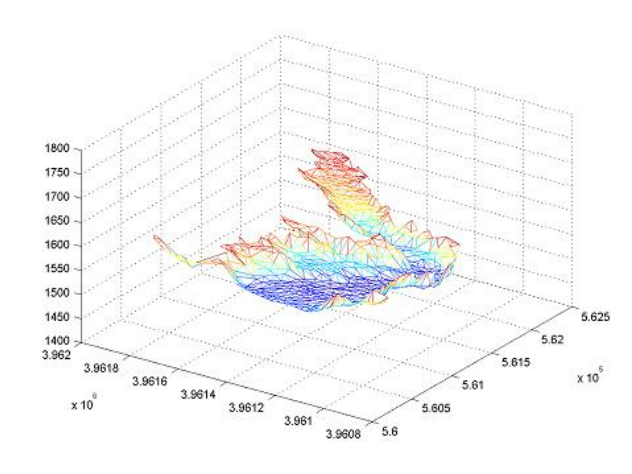
$$V(P) = \sum_{i=0}^{n}\sum_{j=0}^{m_i}(-1)^i V(P_{ij})$$

where P_{ij} means the jth component of the ith level and m_i is the number of components exist in the ith level. Note that this equation subtracts the volumes of the components of the odd levels. To implement this algorithm, the n-simplexes are represented as a list of points. Then, their operations (e.g., convex decomposition, triangulation, volume calculation, etc.) become operations on lists:

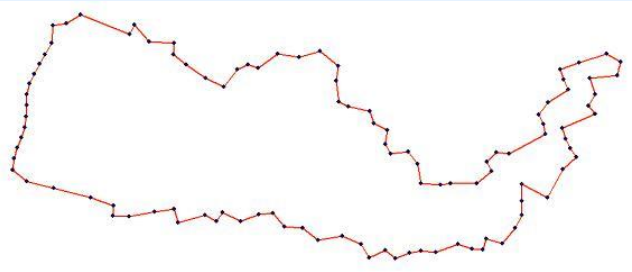
$$\begin{aligned} vS &= 0.5\ (*)\ .\ abs\ .\ det\ .\ map\ (1:) \\ vC &= sum\ .\ map\ vS\ .\ tri \\ vNC &= sum\ (zipWith\ (*)\ (cycle\ [1,(-1)])\ (map\ vC\ cd)) \end{aligned}$$

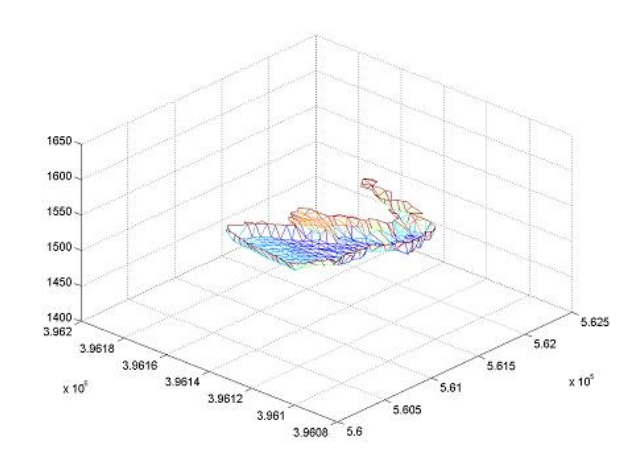
where vS is the volume of an n-simplex, vC is the volume of a convex polytop, vNC is the volume of a non-convex polytop, tri is triangulation of a convex polytop and cd decomposes a non-convex polytop to a set of convex components. Since the representation and operations are defined independent of dimension, the developed algorithms can be used for polytops of any dimension.

The implementation was used to calculate the surface and volume of a lake at certain water levels, which leads to a level-surface-volume diagram. This diagram shows the surface and volume of the lake at different water levels. First, the 3D TIN (Triangulated Irregular Network) of the lake was constructed:

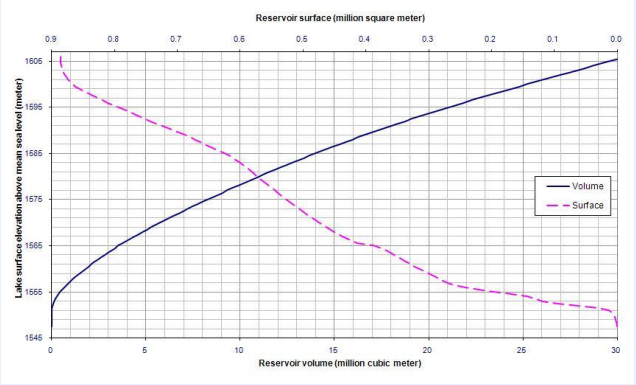


To calculate the surface and volume of the lake at a certain water level, say h, the 3D TIN was intersected with the plan $z = h$, which results in the volume of the lake where $z < h$ and the surface of the lake at $z = h$, whose surface and volume is calculated using the implemented algorithm:





By applying the above process for different water levels, the level-surface-volume diagram was produced:

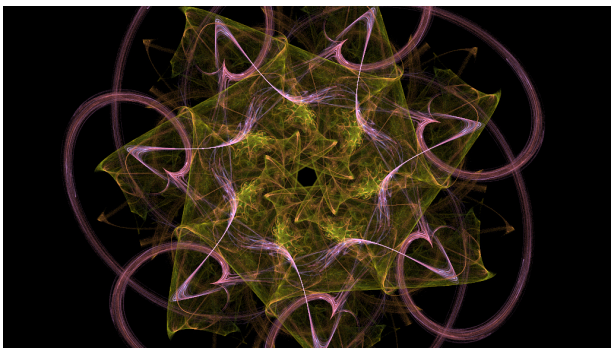


6.6.4 Fl4m6e

Report by:	Claude Heiland-Allen
Status:	unstable

Fl4m6e is a fast fractal flame renderer, currently under active development. Inspired by flam3 and electric sheep, Fl4m6e performs all the rendering calculations on the GPU, using OpenGL shaders. The features so far include runtime generation of low level GLSL source code from abstract scene descriptions, smooth transitions between scenes using rigid transformations, and interactive control of animation and quality settings.

Contrasting to flam3, the aim is not exact image reproducibility, but to get fast enough that pre-rendering videos is not necessary — then a peer-to-peer network exchanging small scene descriptions would supercede a centralized file server and corresponding large bandwidth requirements. However, there is a long way to go before the electric sheep have anything to worry about.



Further reading

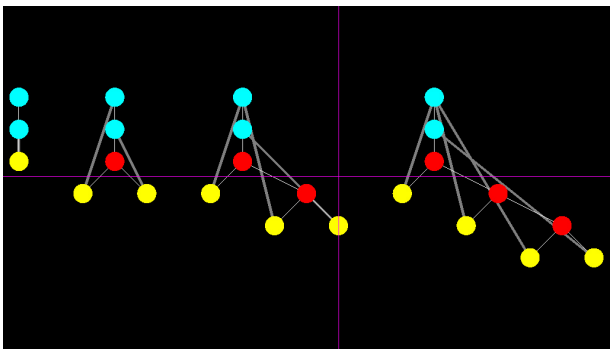
- http://claudiusmaximus.goto10.org/cm/2009-08-28_fl4m6e_proof_of_concept.html
- http://claudiusmaximus.goto10.org/cm/2009-09-24_fl4m6e_in_haskell.html
- http://claudiusmaximus.goto10.org/g/fl4m6e/examples/
- http://flam3.com/
- http://electricsheep.org/

6.6.5 GULCI

Report by:	Claude Heiland-Allen
Status:	unstable

GULCI is a graphical untyped lambda calculus interpreter. Programs are written with mouse clicks and drags, and executed with a keypress. During execution the graph reduction is visualized. GULCI also dumps data on its standard output stream, suitable for sonification. The eventual intent is to use it for a short abstract code performance sometime in the future.

GULCI is the interactive descendant of the non-interactive (but also graphical) ULCIv1, which took the form of audiovisualizations of some simple arithmetical computations in untyped lambda calculus.



Further reading

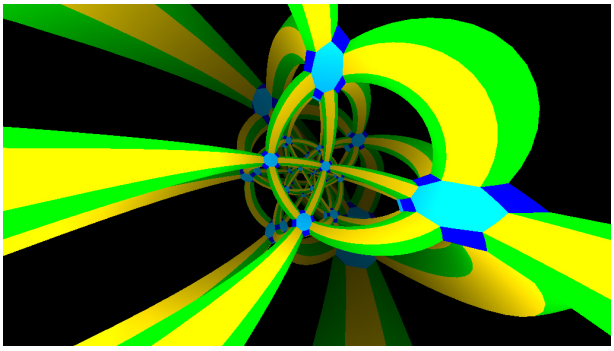
http://claudiusmaximus.goto10.org/cm/2009-06-19_untyped_lambda_calculus_interpretations_v1.html

6.6.6 Reflex

Report by:	Claude Heiland-Allen
Status:	experimental

Reflex is a program to interactively experience variously truncated regular 4D polytopes. The name is inspired by reflection symmetry groups and the eventual intent of using it as an interactive live audiovisual performance environment with quick reactions required, the idea for the software itself is inspired by The Symmetriad.

Starting from the Shlaefli symbol p, q, r, Reflex constructs a 4D symmetry group, from which a wide variety of different forms can be visualized. The projection method results in aesthetically pleasing curves, enhanced by the animation. Currently Reflex accepts commands on its standard input, but the goal is to use a game controller to navigate through the world.



Further reading

- http://claudiusmaximus.goto10.org/cm/2009-10-15_reflex_preview.html
- http://web.mit.edu/~axch/www/Symmetriad/index.html
- http://en.wikipedia.org/wiki/SchlÃďfli_symbol

6.6.7 Citten

Report by:	Stephen Roantree
Status:	alpha, active

Citten brings a new functional language to the GPU shader platform. It is also an investigation into how to create a language that is able to shift target, to some degree, without needing to alter the compiler or language specification. We want to allow the language to define in itself the variable components of the platform. This is done with a view to making the language at least partially resilient to the frequent extensions to shader technologies.

Functional programming suits the nature of traditional shader programs. These programs lack side effects, and are often representations of cascaded formulae. This, combined with Citten's type system, provides programmers with concrete benefits over the existing alternatives.

Citten is currently a first order, strict language. The compiler is being extended to be high order, and to use a more fully featured type system. The current version (0.1.0) is not recommended for general use. This will change with the coming version.

Future plans

In order to lower the barrier of entry, integration with the XNA Content Pipeline is planned.

Providing a more rigorous encapsulation of side effects will be required to elegantly represent certain kinds of operations. This would allow better representation of geometry shaders, and hopefully should allow the coming compute shader stage to be supported with little effort as well.

Further reading

http://github.com/stroan/Citten

6.6.8 Hemkay

Report by:	Patai Gergely
Status:	experimental, active

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

The current version of the player uses the PortAudio bindings for playback, but there is also a yet unreleased functional version based on OpenAL, which puts a much smaller load on the CPU. Also, an OpenGL based graphical frontend is currently in the works.

Further reading

- http://hackage.haskell.org/package/hemkay-core
- http://hackage.haskell.org/package/hemkay
- http://en.wikipedia.org/wiki/MOD_(file_format)

6.7 Proof Assistants and Reasoning

6.7.1 HTab

Report by:	Guillaume Hoffmann
Participants:	Carlos Areces, Daniel Gorin
Status:	active development
Current release:	1.5.3

HTab is an automated theorem prover for hybrid logics (http://haskell.org/communities/11-2009/html/report.html#sect6.7.3) based on a tableau calculus. It handles hybrid logic with nominals, satisfaction operators, converse modalities, universal and difference modalities, the down-arrow binder, and role inclusion.

The source code is distributed under the terms of the GNU GPL.

Further reading

- http://code.google.com/p/intohylo/

6.7.2 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.

Recently some minor deficiencies of Haskabelle have been amended. The tool chain has been presented at the PEPM'10 workshop with a workshop paper.

Further reading

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

6.7.3 Plastic

Report by:	Robin Adams
Participants:	Zhaohui Luo
Status:	prototype

The Plastic proof assistant was developed by Paul Callaghan in 2001 as an implementation of the logical framework LF, a Church-typed version of Martin-Löf's logical framework. Its development never advanced far beyond the experimental, prototype stage.

We have recently taken up the development of Plastic. A few years ago, Callaghan kindly adapted plastic to implement the Type Theoretic Framework, a framework for declaring several Logic-Enriched Type Theories (LETTs). We have already used this modified version of Plastic to formalize Weyl's *Das Kontinuum* in a classical predicative LETT.

We are currently experimenting with using Plastic for carrying out *pluralist formalizations*, where work in one mathematical setting may be reused in another setting, by providing an appropriate translation between the two.

For example, we have a proof script that proves a theorem A in a classical LETT. We may reuse this in a constructive LETT by plugging in a module that describes the double negation translation. The result is a proof of the double negation translation of A in the constructive LETT.

Plastic is written in Haskell.

Further reading

Details about this project will appear here soon: http://www.cs.rhul.ac.uk/~robin/plastic

6.7.4 Free Theorems for Haskell

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

Free theorems are statements about program behavior derived from (polymorphic) types. Their origin is the polymorphic lambda-calculus, but they have also been applied to programs in more realistic languages like Haskell. Since there is a semantic gap between the original calculus and modern functional languages, the underlying theory (of relational parametricity) needs to be refined and extended. We aim to provide such new theoretical foundations, as well as to apply the theoretical results to practical problems. A recent publication is "Automatically Generating Counterexamples to Naive Free Theorems" (FLOPS'10).

On the practical side, we maintain a library and tools for generating free theorems from Haskell types, originally implemented by Sascha Böhme and with contributions from Joachim Breitner and now Matthias Bartsch. Both the library and a shell-based tool are available from Hackage (as free-theorems and ft-shell, respectively). There is also a web-based tool at http://www-ps.iai.uni-bonn.de/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

A new version of the web-based tool will be online very soon, which will enable the user to declare their own algebraic data types and so on, and then to derive free theorems from types involving those. (Previously, this was only possible in the shell-based tool.) Also, in addition to plain text, LaTeX source, and PDF output, the new version will be able to output inline graphics with nicely typeset theorems.

Further reading

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

6.7.5 CSP-M animator and model checker

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

We develop a Haskell based, integrated CSP-M animator and model checker.

Communicating-Sequential-Processes is a formalism for concurrent systems, invented by Tony Hoare.

Our Haskell-CSP-Tool features:

- FDR compatibility
- Fast computation of state spaces
- GTK+ based graphical user interface
- Support for shared-memory-parallelism / multicore CPUs

Binary releases of the gui-tool are available for download via http://www.stups.uni-duesseldorf.de/~fontaine/csp.

The aim of the project is not only to write a black-box end-user tool, but also to provide components that can be useful for other formal methods researchers who are investigating communicating sequential processes.

The following packages are available on Hackage:

CSPM-Frontend A FDR compatible CSP-M parser.

CSPM-CoreLanguage An abstract interface for a CSP core language.

CSPM-FiringRules An implementation of the firing rule semantic of CSP.

CSPM-Interpreter An interpreter for the functional sub language included in FDR.

CSPM-cspm A small command line executable that demonstrates how to clue the above libraries together.

Further reading

http://www.stups.uni-duesseldorf.de/~fontaine/csp

6.8 Hardware Design

6.8.1 ForSyDe

Report by:	Ingo Sander
Participants:	Hosein Attarzadeh, 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.

ForSyDe allows to model heterogeneous embedded systems at a high level of abstraction by providing libraries for different models of computation (MoCs). This allows to model systems consisting of both digital and analog hardware.

The current release is ForSyDe 3.1, which contains two implementations of ForSyDe. The shallow-embedded DSL has been designed for the modeling purpose and provides a rapid-prototyping framework which allows to model and simulate heterogeneous embedded systems based on different MoCs. The deep-embedded DSL supports only the synchronous MoC,

but comes with an embedded compiler with different backends (simulation, synthesizable VHDL and GraphML). It is possible to integrate and simulate shallow-embedded models with deep-embedded models.

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

Features

ForSyDe systems are modeled as concurrent process networks, where processes communicate via signals. To create processes, ForSyDe uses higher-order functions to implement the concept of process constructors, which leads to a structured model with a clear separation of computation from communication.

The two DSL flavors of ForSyDe offer different features:

1. 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. At present ForSyDe supports the following models of computation.

   - Synchronous MoC
   - Untimed MoC
   - Continuous Time MoC

   Process networks belonging to different MoCs communicate via domain interfaces, which establish a relation with respect to timing between two MoCs.

2. Deep-embedded DSL

   Deep-embedded signals (`ForSyDe.Signal`), based on the same concepts as Lava (→ 8.5), 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, specification of behavior is 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 hierarchy by component instantiation
   - Support for fixed-sized vectors

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 and digital hardware, 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/

6.8.2 Kansas Lava

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

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

Driven by a self-imposed requirement to implementing some specific telemetry circuits in Lava, we have made a number of recent improvements to both the external API and the internal representations used. We have retained our dual shallow/deep representation of signals in general, but now have a number of externally visible abstractions for combinatorial, sequential, and enabled signals. We also have new abstractions for memory and memory updates. Internally, we found the need to represent unknown values inside our circuits, so we made aggressive use of type functions to lift our values in a principled and regular way. This design decision unfortunately complicates the internals of Kansas Lava, but the external API remains unaffected.

An overarching design decision is the aggressive use of an algebra over *commutable functors and observable functors* for circuit refinement, the details of which we hope to write up this summer.

We have also been working on a new debugging system, which combines the deep and shallow embedding in a way to allow probes to be inserted onto functions. The values of the usage of these functions can be observed, as well as used to generate test vectors.

A release is planned for late summer, and will be available on Hackage. Recent presentations about Kansas Lava include (slides on website):

- May 18th, What's the matter with Kansas Lava?, Eleventh Symposium on Trends in Functional Programming, Norman, OK.

- May 18th, The Internals and Externals of Kansas Lava, Eleventh Symposium on Trends in Functional Programming, Norman, OK.

- May 11th, Generating Implementations of Error Correcting Codes using Kansas Lava, 10th Annual High Confidence Software and Systems Conference, Linthicum Heights, MD.

- February 26th, Forward Error Correction Codes and Kansas Lava, Brigham Young University, Provo, Utah.

Further reading

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

6.9 Natural Language Processing

6.9.1 NLP

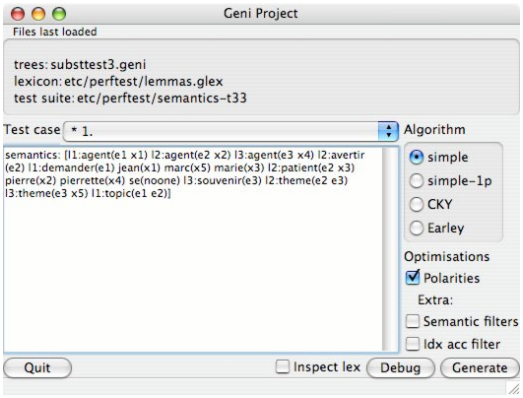
Report by:	Eric Kow

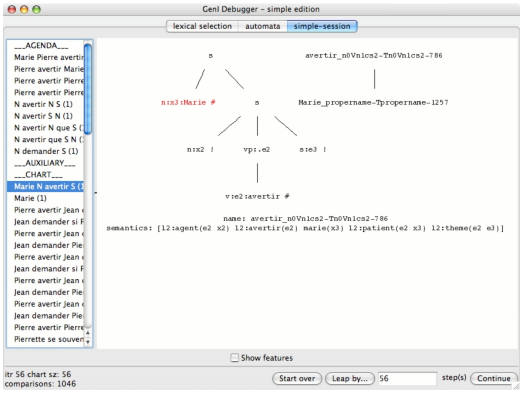
See: http://haskell.org/communities/05-2009/html/report.html#sect6.10.1.

6.9.2 GenI

Report by:	Eric Kow

GenI is a surface realizer for Tree Adjoining Grammars. Surface realization can be seen a subtask of natural language generation (producing natural language utterances, e.g., English texts, out of abstract inputs). GenI in particular takes 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 0.20.1 (2009-10-01), which offers cleaner interactions with the third-party tools (using JSON), simpler installation on MacOS X and a user manual. For more information, please contact us on the geni-users mailing list.

Further reading

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

6.9.3 Grammatical Framework

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

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

An important objective in the language development was to make it possible to develop modular grammars. The language provides modular system inspired from ML but adapted to the specific requirements in GF. The modules system was exploited to a large extent in the Resource Libraries project. The library provides large linguistically motivated grammars for a number of languages. When the languages are closely related the common parts in the grammar could be shared using the modules system. Currently there are complete grammars for Bulgarian, Danish, English, Finnish, French, German, Interlingua, Italian, Norwegian, Russian, Spanish, and Swedish. There are work in progress grammars 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.

On 24 June 2009 after one year of hard work the previous beta version of GF 3.0 is turned into stable release. 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 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.

We are continuing to work hard. Some of the projects that keeps us busy:

- Type checker for dependent types in the interpreter. Currently only the compiler has type checker.
- Semantic parsing — i.e., parser which considers not only the syntax but also the semantics of the language. The semantics is encoded using dependent types.
- New resource grammars for Romanian and Polish
- Visualization of parse trees and dependency trees in addition to syntax trees
- Experiments with natural languages and big ontologies
- Building wide coverage Swedish grammar based on the resource grammar.

Further reading

www.digitalgrammars.com/gf

6.10 Bioinformatics

6.10.1 Bein

Report by:	Fred Ross
Status:	preparing for first release and deployment

Scientists doing exploratory data analysis typically run short sequences of commands with a range of parameters on a small number of data sets. In the standard unix shell, this results in a plethora of files with no obvious description of how they were created. Full workflow managers such as Knime excel for long, complicated sequences to be run over and over again on a large number of data sets, but are useless for exploration.

Bein fills this niche. It tracks programs, files (and in later releases biological sequence data, since it is being written for a bioinformatics group), and executions of programs on given sets of inputs, interfaces with LSF clusters, and exposes a web interface to users.

The first release and deployment in Lausanne is scheduled for late May, 2010. It will be deployed elsewhere in Switzerland under the auspices of the SyBit project. The code, and all future updates, are available from GitHub.

Further reading

http://github.com/madhadron/bein

6.10.2 Biohaskell (previously: Bioinformatics tools)

Report by:	Ketil Malde

The Haskell bioinformatics library supports working with nucleotide and protein sequences and associated data. It supports a variety of file and alignment formats, and provides basic functions for working with sequences.

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 it comes with a QuickCheck test and benchmarking suite.

Recent changes include extensive features for handling the native Roche 454 sequence formats (flowgrams), including quality filtering and trimming.

The library has been used in a number of applications, the latest are are *flowsim*, a simulator for 454-style sequences, and *flowt*, a fast filter for removing duplicate clones.

Further reading

- http://blog.malde.org/
- darcs repositories at http://malde.org/~ketil/biohaskell

6.11 Games

6.11.1 Freekick2

Report by:	Antti Salonen
Status:	experimental, active development

Freekick2 is a 2D arcade-style soccer game, written in Haskell. It is still very young, but playable. It features texture-mapped graphics, a simple but functional and well playing AI, and the ability to import Sensible Soccer team data files. Freekick2 uses the Haskell bindings to OpenGL, FTGL and SDL for input handling, graphics and GUI. It is available at Hackage. Future plans include improving the AI and the gameplay.

Further reading

- http://github.com/anttisalonen/freekick2
- http://codeflow.wordpress.com/2010/05/04/announcing-freekick2/

6.11.2 Dungeons of Wor

Report by:	Patai Gergely
Status:	experimental, active

Dungeons of Wor is an homage to the classic arcade game, Wizard of Wor. It uses the artwork and levels from the arcade version, but the gameplay mechanics differ from the original in several ways.

This game is also an experiment in functional reactive programming, so it might be a useful resource to anyone interested in this topic. It was coded using the Simple variant of the experimental Elerea library (→ 6.5.2), which provides discrete streams as first-class values.

Further reading

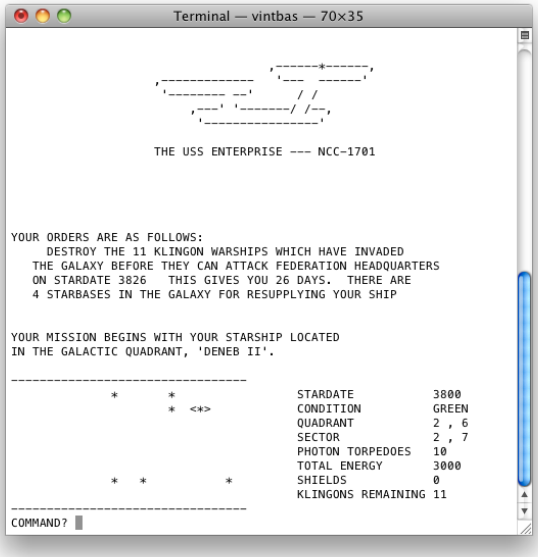
- http://hackage.haskell.org/package/dow
- http://en.wikipedia.org/wiki/Wizard_of_Wor

6.12 Programming Languages

6.12.1 Vintage BASIC

Report by:	Lyle Kopnicky
Current release:	1.0.1
Portability:	GHC 6.10, 6.12

Vintage BASIC is an interpreter for microcomputer-era BASIC, written in Haskell. It is fully unit-tested, and implements all common features of the language. The web site includes games from Creative Computing's BASIC Computer Games, all of which can be run under the interpreter.



The interpreter makes use of a novel technique for implementing BASIC's dynamic control structures: resumable exceptions. For example, in BASIC loops, the FOR keyword becomes an exception handler, and the NEXT keyword throws an exception. Furthermore, the exceptions are caught in the continuation, rather than the containing expression. The handlers can also be selectively persisted after handling exceptions. Because of these two features, I refer to them as "durable traps". The DurableTraps library is fully abstracted using monad transformers, and can be used in any program.

Further reading

http://www.vintage-basic.net

6.12.2 LQPL — A quantum programming language compiler and emulator

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

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

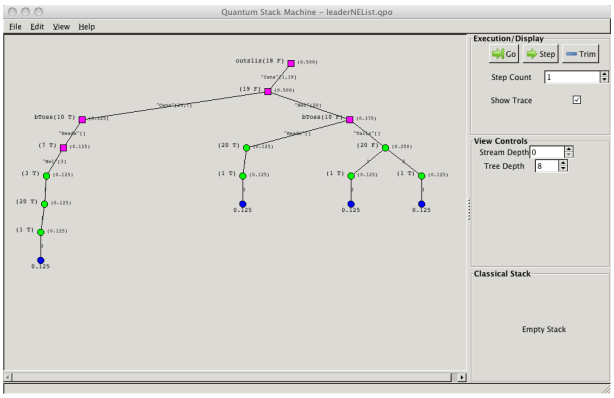
This programming language was inspired by Peter Selinger's paper "Toward a Quantum Programming Language". LQPL incorporates a simplified module / include system (more like C's include than Haskell's import), predefined unitary transforms, quantum control and classical control, algebraic data types, and operations on purely classical data. The compiler translates LQPL programs into an assembler language targeted to a quantum stack machine. The emulator, written using Gtk2Hs, translates the assembler to machine code and provides visualization of the program as it executes.

In version 0.8.4, some examples of probabilistic programming have been added and the operational semantics of `measure`, `use`, and `case` have been revised slightly to make them show more sensibly in the associated emulator.

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

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

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



Further reading

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

6.13 Others

6.13.1 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))
```

IgorII has been extended to use catamorphisms on lists as higher-order templates. After enabling the higher-order mode, given the previous examples of `reverse`, the system outputs the following solution:

```
reverse xs    = foldr snoc [] xs

snoc x xs     = foldr cons [x] xs
cons x (y:ys) = x:(y:ys)
```

Recently, the use of list-catamorphism has been generalized to arbitrary inductive data types. Based on the Pointless Haskell library IgorII uses its generic implementation of recursion patterns to solve recursive problems as e.g. *mirroring binary trees*, *computing the power set*, or to find a recursive solution for the *towers of hanoi*, to mention just a few.

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
- using catamorphisms on arbitrary inductive data types as higher-order templates or generic recursion schemes

Current Status and Future Plans

The original version of IgorII is implemented in the reflective rewriting based programming and specification language Maude. However, a Haskell implementation of the algorithm is the current research prototype. Both can be obtained from the project page.

A tool demo and a research paper about the use of catamorphisms as higher-order templates were presented at PEPM 2010.

For the future, we plan to extend the system to use other type morphisms as generic recursion schemes. Also it would be worth investigating to which extent knowledge about types, e.g. universal properties, can be used for the synthesis process, e.g. to guide the search or resolve ambiguities.

Further reading

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

6.13.2 Yogurt

Report by:	Martijn van Steenbergen

See: http://haskell.org/communities/05-2009/html/report.html#sect6.12.6.

6.13.3 Bullet

Report by:	Csaba Hruska
Status:	experimental, active development

Bullet is a professional open source multi-threaded 3D Collision Detection and Rigid Body Dynamics Library written in C++. It is free for commercial use under the zlib license. The Haskell bindings ship their own C compatibility layer, so the library can be used without modifications.

At the current state of the project only basic services are accessible from Haskell, i.e., you can load collision shapes and step the simulation. More advanced Bullet features (constraints, soft body simulation etc.) will be added later.

Further reading

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

6.13.4 arbtt

Report by:	Joachim Breitner
Status:	working

The program arbtt, the automatic rule-based time tracker, allows you to investigate how you spend your time, without having to manually specify what you are doing. arbtt records what windows are open and active, and provides you with a powerful rule-based language to afterwards categorize your work. And it comes with documentation!

Since the last HCAR, arbtt was added to Debian. At the Haskell Hackathon in Zürich, three new contributors to arbtt joined the development.

Further reading

- http://www.joachim-breitner.de/projects#arbtt
  http://www.joachim-breitner.de/blog/archives/336-The-Automatic-Rule-Based-Time-Tracker.html
- http://darcs.nomeata.de/arbtt/doc/users_guide/

6.13.5 uacpid

Report by:	Dino Morelli
Status:	experimental, actively developed

uacpid is a daemon designed to be run in userspace that will monitor the local system's acpid socket for hardware events. These events can then be acted upon by handlers with access to the user's environment. Configuration of uacpid closely mimics that of acpid.

uacpid is available in binary form for Arch Linux through the AUR and can be acquired using darcs or other methods.

Further reading

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

6.13.6 cltw (Twitter API command-line utility)

Report by:	Dino Morelli
Status:	experimental, actively developed

This is a tool for performing some Twitter API functions from the command-line. So far supporting three calls: statuses/followers, statuses/friends, statuses/update.

cltw is available from Hackage, the darcs repository below, and also in binary form for Arch Linux through the AUR.

Further reading

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

7 Commercial Users

7.1 Well-Typed LLP

Report by:	Ian Lynagh
Participants:	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⟩.

This has been another busy 6 months for us, with a mixture of proprietary contracts and open source work. We have a number of interesting opportunities on the horizon, and are looking forward to the next 6 months.

In the coming weeks we will begin a 2-year project, funded by Microsoft Research, to push the real-world adoption and practical development of parallel Haskell with GHC. We are currently seeking organizations to take part. For more details, see our blog.

As the business grows, we are also looking for more people. If you are interested in a Haskell job, see our blog for details.

Further reading

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

7.2 Bluespec tools for design of complex chips and hardware accelerators

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. Unlike HW design with C, which can only be used for "loop-and-array" computations, BSV is universal, accommodating the diverse range of blocks found in modern 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. The core tool is implemented in Haskell (well over 100K lines).

In addition to the core synthesis tool, Bluespec provides a fast simulation tool for BSV, and extensive libraries and infrastructure to make it easy to build FPGA-based accelerators for computationally intensive software, including for the Xilinx XUP board popular in universities.

These industrial strength tools have enabled some large designs (over a million gates) and significant architecture research projects in academia and industry, because complex architectural models can now be coded with the same convenience of expression as SW languages, but with the execution speed of FPGAs.

Status and availability

BSV tools, available since 2004, are in use by several major semiconductor and electronic equipment companies, and universities. The tools are free for academic teaching and research.

Further reading

- R.S.Nikhil, *Bluespec, a General-Purpose Approach to High-Level Synthesis Based on Parallel Atomic Transactions*, in *High Level Synthesis: from Algorithm to Digital Circuit, Philippe Coussy and Adam Morawiec (editors)*, Springer, 2008, pp. 129-146.
- Small illustrative examples: http://sites.google.com/a/bluespec.com/learning-bluespec/Home/Small-Examples
- 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

7.3 Industrial Haskell Group

Report by:	Duncan Coutts
Participants:	Ian Lynagh

The Industrial Haskell Group (IHG) is an organization to support the needs of commercial users of Haskell. It was formed in early 2009, and has already made a significant contribution to the recent GHC 6.12 release: In the first 6 months collaborative development scheme, the IHG has funded work on dynamic libraries, more flexible Integer library support for GHC, and Cabal development work. The details of these projects are on the website.

While the aim had been to start the next round of the scheme at the beginning of this year, it has been on hiatus due to Duncan taking longer than expected to complete his PhD thesis. Well-Typed (→ 7.1) is however increasing its capacity and the plan is to run the collaborative development scheme on a continuous basis in future.

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

Further reading

http://industry.haskell.org/

7.4 typLAB

Report by:	Sebastiaan Visser
Participants:	Lon Boonen, Erik Hesselink, Salar al Khafaji

TypLAB is a startup company located in the city center of Amsterdam. We investigate and develop new ways of creating and consuming online content. Our current focus is in building an online environment in which users can manage content in unconventional ways.

The Happstack powered server application, the automated deployment scripts, the JavaScript preprocessors; all code running at the server is written entirely in Haskell. The vast amount of Haskell packages, especially for XML manipulation and generic programming, allow us to easily interface with our Berkeley XML database back-end. A large part of our application is written in JavaScript and runs in the client. Most of the JavaScript code is heavily inspired by functional (reactive) programming and enables us to achieve a very high level of abstraction, even in the web browser.

TypLAB is showing that combining the theoretical foundations of computer science with the day-to-day practice of the web allows for building high-quality web-applications. Still a lot of work has to be done before our first beta will see the light. Please keep in touch with our progress by checking out our weblog.

Further reading

- http://typlab.com
- http://blog.typlab.com

7.5 factis research GmbH

Report by:	Stefan Wehr
Participants:	David Leuschner, Harald Fischer
Status:	beta, active development

factis research, located in Freiburg, Germany, develops reliable and user-friendly mobile solutions. Our client software runs under J2ME, Symbian, iPhone OS, Android, and Blackberry. The server components are implemented in Python and Haskell.

We are actively using Haskell for a number of projects, most of which are released under an open-source license:

- Server backends for our mobile software solutions.
- *DPM* (→ 4.3.5), a patch manager for darcs.
- *HTF* (→ 4.2.1), a test framework.
- *fos* (http://openfactis.org/fos/), a customer relationship management tool. Originally, fos was written as a Haskell GTK application, but we are currently rewriting it as a web application.
- *ntee* (http://openfactis.org/ntee), an adaptation of the Unix tool `tee` to network streams.

Further reading

http://www.factisresearch.com/

7.6 Tsuru Capital

Report by:	Simon Cranshaw

Tsuru Capital is engaged in high-frequency market-making on options markets. Tsuru is a private company, and trades with its own capital. Tsuru Capital currently runs arbitrage based liquidity provision

strategies on the Kospi 200 index and plans to expand to Nikkei 225 index, and other electronic markets, over the next year.

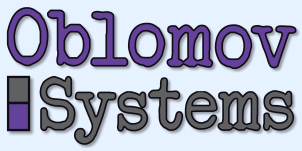
The trading software has been developed entirely in Haskell, and is one of the few systems in the world written completely in a functional language.

Further reading

http://tsurucapital.com/en/

7.7 Oblomov Systems

Report by: Martijn Schrage



Oblomov Systems is a one-person software company based in Utrecht, The Netherlands. Founded in 2009 for the Proxima 2.0 project (→ 6.4.5), Oblomov has since then been working on a number of Haskell-related projects. The main focus lies on web-applications and (web-based) editors. Haskell has turned out to be extremely useful for implementing web servers that communicate with JavaScript clients or iPhone apps.

Awaiting the acceptance of Haskell by the world at large, Oblomov Systems also offers software solutions in Java, Objective C, and C#, as well as on the iPhone.

Further reading

http://www.oblomov.com

8 Research and User Groups

8.1 Artificial Intelligence and Software Technology at Goethe-University Frankfurt

Report by: David Sabel
Participants: Conrad Rau, Manfred Schmidt-Schauß

One of our research topics focuses on programming language semantics, especially on contextual equivalence which is usually based on the operational semantics of the language. We explored several call-by-need lambda calculi. Deterministic call-by-need lambda calculi with letrec provide a semantics for the core language of Haskell. In the setting of such an extended calculus we proved correctness of strictness analysis using abstract reduction. Furthermore, we proved equivalence of the call-by-name and call-by-need semantics of an extended lambda calculus with `letrec`, `case`, and constructors.

Recently, we extended our investigations to parametric polymorphism and and showed correctness of type dependent program transformations. Most recently, in collaboration with *Elena Machkasova* we have shown that the call-by-need lambda calculus with letrec is isomorphic to the lazy lambda calculus and that bisimilarity coincides with contextual equivalence in the call-by-need lambda calculus with letrec.

We also 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` in Haskell. We investigated a call-by-need lambda calculus extended with McCarthy's `amb` and an abstract machine for lazy evaluation of concurrent computations. We have shown that mutual similarity is a sound proof method w.r.t. contextual equivalence in a class of untyped higher-order nondeterministic call-by-need lambda calculi.

In a recently started research project we try to automatize correctness proofs of program transformations. A main step for this goal is the computation of overlappings between reductions of the operational semantics and transformations steps. This computation requires the combination of several unification algorithms. We implemented a prototype of this combined algorithm in Haskell.

As a further research topic 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.

Further reading

http://www.ki.informatik.uni-frankfurt.de/research/HCAR.html

8.2 Functional Programming at the University of Kent

Report by: Olaf Chitil

The Functional Programming group at Kent is a subgroup of the newly formed Programming Languages and Systems Group of the School of Computing. We are a group of staff and students with shared interests in functional programming. While our work is not limited to Haskell — in particular our interest in Erlang has been growing — Haskell provides a major focus and common language for teaching and research.

Our members pursue a variety of Haskell-related projects, some of which are reported in other sections of this report. Thomas Schilling is developing ideas for improving type error messages for GHC. Part of this work is currently under review for ICFP and will be included in the next major release of the Scion IDE library. Neil Brown developed a library for Communicating Haskell Processes (CHP). Several members develop an occam compiler in Haskell (Tock).

Further reading

- PLAS group: http://www.cs.kent.ac.uk/research/groups/plas/
- Refactoring Functional Programs: http://www.cs.kent.ac.uk/research/groups/plas/hare.html
- Tracing and debugging with Hat: http://www.haskell.org/hat
- Heat: http://www.cs.kent.ac.uk/projects/heat/
- Scion: http://code.google.com/p/scion-lib/
- Message-Passing Concurrency for Haskell using the CHP library: http://chplib.wordpress.com/
- Tock: http://projects.cs.kent.ac.uk/projects/tock/

8.3 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

See: http://haskell.org/communities/05-2009/html/report.html#sect8.5.

8.4 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 Haskell-related projects are:

- *The Monad Zipper*: Limitations of monad stacks get in the way of developing highly modular programs with effects. This pearl demonstrates that Functional Programming's abstraction tools are up to the challenge. Of course, abstraction must be followed by clever instantiation: Huet's zipper for the monad stack makes components jump through unanticipated hoops. This is joint work with Bruno Oliveira.

- *EffectiveAdvice:* EffectiveAdvice is a disciplined model of (AOP-style) advice, inspired by Aldrich's Open Modules, that has full support for effects in both base components and advice. EffectiveAdvice is implemented as a Haskell library. Advice is modeled by mixin inheritance and effects are modeled by monads. Interference patterns previously identified in the literature are expressed as combinators. Equivalence of advice, as well as base components, can be checked by equational reasoning. Parametricity, together with the combinators, is used to prove two harmless advice theorems. The result is an effective model of advice that supports effects in both advice and base components, and allows these effects to be separated with strong non-interference guarantees, or merged as needed. This is joint work with Bruno Oliveira and William Cook.

- *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, Louis-Julien Guillemette and Dominic Orchard.

- *The Monadic Constraint Programming Framework:* The main article on the MCP framework by Tom Schrijvers, Peter Stuckey and Phil Wadler has appeared in the Journal of Functional Programming. 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 finite domain 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++). Models can be compiled to C++ code, can be solved by calling Gecode from Haskell at runtime, or can be solved purely in Haskell itself.

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/

8.5 Functional Programming at Chalmers

Report by:	Jean-Philippe Bernardy

Functional Programming is an important component of the Department of Computer Science and Engineering at Chalmers. In particular, Haskell has a very important place, as it is used as the vehicle for teaching and numerous projects. Besides functional programming, language technology, and in particular domain specific languages is a common aspect in our projects.

The Functional Programming research group has 5 faculty members and 12 postdoc and doctoral students. Research is going on in various exciting topics:

Property-based testing QuickCheck is the basis for a European Union project on Property Based Testing (www.protest-project.eu). We are applying the QuickCheck approach to Erlang software, together with Ericsson, Quviq, and others. Much recent work has focused on PULSE, the ProTest User-Level Scheduler for Erlang, which has been used to find race conditions in industrial software — see our ICFP 2009 paper for details. A new tool, QuickSpec, generates algebraic specifications for an API automatically, in the form of equations verified by random testing. We will publish about it at TAP 2010; an earlier paper can be found here: http://www.cse.chalmers.se/~nicsma/quickspec.pdf. Lastly, we have devised a technique to speed up testing of polymor-

phic properties: http://publications.lib.chalmers.se/cpl/record/index.xsql?pubid=99387.

Natural language technology Grammatical Framework (→ 6.9.3) is a declarative language for describing natural language grammars. It is useful in various applications ranging from natural language generation, parsing and translation to software localization. The framework provides a library of large coverage grammars for currently fifteen languages from which the developers could derive smaller grammars specific for the semantics of a particular application.

Generic Programming Starting with Polytypic Programming in 1995 there is a long history of generic programming research at Chalmers. Recent developments include work on dependent types (a JFP paper + library around "Algebra of Programming using Agda"; as well as an account of parametricity for dependent types), two survey papers "C++ Concepts =? Haskell Type Classes" and "Comparing GP Libs in Haskell" and applications to sustainable development with the Potsdam Institute for Climate Impact Research (http://www.pik-potsdam.de/. Patrik Jansson (with Sibylle Schupp) chaired the last Workshop on Generic Programming. Related publications are available here: http://publications.lib.chalmers.se/cpl/lists/publications/people/html/index.xsql?ids=701&lyear=1900&hyear=2020

Language-based security SecLib is a light-weight library to provide security policies for Haskell programs. The library provides means to preserve confidentiality of data (i.e., secret information is not leaked) as well as the ability to express intended releases of information known as declassification. Besides confidentiality policies, the library also supports another important aspect of security: integrity of data. SecLib provides an attractive, intuitive, and simple setting to explore the security policies needed by real programs.

Type theory Type theory is strongly connected to functional programming research. Many dependently-typed programming languages and type-based proof assistants have been developed at Chalmers. The Agda system (→ 3.2.2) is the latest in this line, and is of particular interest to Haskell programmers. We encourage you to experiment with programs and proofs in Agda as a "dependently typed Haskell".

DSP programming Feldspar is a domain-specific language for digital signal processing (DSP), developed in co-operation by Ericsson, Chalmers FP group and Eötvös Loránd (ELTE) University in Budapest. The motivating application is telecom processing, but the language is intended to be more general. As a first stage, we have focused on the data-intensive numeric algorithms which are at the core of any DSP application. More recently, we have started to work on extending the language to deal with more system-level aspects. The data processing language is purely functional and highly inspired by Haskell. Currently the language is implemented as an embedded language in Haskell.

The implementation is available from Hackage: http://hackage.haskell.org/package/feldspar-language. There is also a code generator, developed at ELTE University: http://hackage.haskell.org/package/feldspar-compiler

See also the official project page: http://dsl4dsp.inf.elte.hu

Hardware design/verification The functional programming group has developed three different hardware description languages — Lava, Wired and Chalk (chronological order) — implemented in Haskell. Each language targets a different abstraction level. The basic idea behind all three is to model circuits as functions from inputs to outputs. This allows structural hardware description in standard functional programming style.

Chalk is a new language for architecture design. Once you have defined a Chalk circuit, you can simulate it, or explore it further using non-standard interpretations. This is particularly useful if you want to perform high-level power and performance analysis early on in the design process.

More info: http://www.cse.chalmers.se/~wouter/Publications/DCC2010.pdf

In **Lava**, circuits are described at the gate level (with some RTL support). The version developed at Chalmers (http://www.cse.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.

Wired is an extension to Lava, targeting (not exclusively) semi-custom VLSI design. A particular aim of Wired is to give the designer more control over on-chip wires' effects on performance. Some features of Wired are:

- Initial description can be purely functional (a la Lava).
- Incremental specification of physical aspects.
- Accurate, wire-aware timing/power analysis within the system.
- Support for an academic 45nm cell library.

Wired is not actively developed at the moment, but the system has recently been used to explore the layout

of multipliers (Kasyab P. Subramaniyan, Emil Axelsson, Mary Sheeran and Per Larsson-Edefors. Layout Exploration of Geometrically Accurate Arithmetic Circuits. *Proceedings of IEEE International Conference of Electronics, Circuits and Systems.* 2009).

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

Automated reasoning Equinox is an automated theorem prover for pure first-order logic with equality. Equinox actually implements a hierarchy of logics, realized as a stack of theorem provers that use abstraction refinement to talk with each other. In the bottom sits an efficient SAT solver. Paradox is a finite-domain model finder for pure first-order logic with equality. Paradox is a MACE-style model finder, which means that it translates a first-order problem into a sequence of SAT problems, which are solved by a SAT solver. Infinox is an automated tool for analyzing first-order logic problems, aimed at showing finite unsatisfiability, i.e. the absence of models with finite domains. All three tools are developed in Haskell.

Teaching Haskell is present in the curriculum as early as the first year of the Bachelors program. We have three courses solely dedicated to functional programming (of which two are Masters-level courses), but we also provide courses which use Haskell for teaching other aspects of computer science, such as programming languages, compiler construction, hardware description and verification, data structures and programming paradigms.

Student Projects Masters students Anders Mortberg and Bassel Mannaa are implementing basic computer algorithms in Haskell. Anders is representing the solutions of linear systems of equations over a coherent ring and Bassel is representing the algebraic closure of a field and Newton's solution of polynomial equations with Puiseux series.

8.6 Dutch Haskell User Group

Report by: Tom Lokhorst

The Dutch Haskell User Group is a diverse group of people interested in Haskell and functional programming.

Since the inception of our user group in April of 2009, we have had monthly meetings and an afternoon symposium. Our meetings alternate between pure socializing and evenings that include talks by members.

We have a wide range of international members; people using functional programming in academia, as a hobby, or for commercial purposes.

Anyone is welcome to join, from beginners to advanced users. Do join us!

Further reading

http://dutchhug.nl/

8.7 San Simón Haskell Community

Report by: Carlos Gomez

The San Simón Haskell Community from San Simón University Cochabamba-Bolivia, is an informal Spanish group that search to learn, share information, knowledge and experience related to the functional paradigm.

Since more than a year, we are trying to expand our community all across Latin American Haskell programmers, and in order to do that, we created a web page (http://comunidadhaskell.org) that serves us as a medium of communication and work environment. All Haskell programmers are welcome to contribute to this site.

Our main activity is the development of projects, and related to that we have information links, a wiki, a blog, some materials and lately we have a section for challenges related to Haskell (http://challenges.comunidadhaskell.org). We started an event for every year in which we present the projects of the last year. On 15th April 2010, we celebrated our 1st Open House Haskell Community in which we presented our projects.

You can also meet us on Facebook, this community is open to all Haskell programmers and specially to Spanish Haskell programmers.

Further reading

http://comunidadhaskell.org

8.8 Functional Programming at KU

Report by: Andy Gill
Status: ongoing

Functional Programming remains active at KU and the Computer Systems Design Laboratory in ITTC. The System Level Design Group (led by Perry Alexander) and the Functional Programming Group (led by Andy Gill) together form the core functional programming initiative at KU. Apart from Kansas Lava (→ 6.8.2) and ChalkBoard (→ 5.10.4), there are many other FP and Haskell related things going on.

- We are developing a Haskell version of HOL. Traditionally, members of the higher-order logic (HOL) theorem proving family have been implemented in the Standard ML programming language or one of its derivatives. HaskHOL aims to break with tradition by implementing a lightweight HOL theorem prover library as a Haskell hosted domain specific language. Based on the HOL Light logical system, HaskHOL aims to provide the ability for Haskell users to reason about their code directly without having to transform it or otherwise export it to an external tool. For details talk to Evan Austin.

- We are actively working on enabling *Type-Directed Specification Refinement in Rosetta*. Rosetta is a specification language that focuses on the interaction between different domains, such as state-based and signal-based domains. With dependent types, first-class types, and reflection, there are many areas where a traditional all-or-nothing typing analysis would be impractical — especially when considering that specifications are likely written at first in a high-level, incomplete fashion. This project uses InterpreterLib (http://haskell.org/communities/11-2008/html/report.html#sect5.5.6) and various Rosetta analysis tools to define a typing analysis that attempts to extract typing information, constraints, and errors to present to the user, in order to guide the specification refinement process. It is in the early stages of development, but may eventually link up with HaskHOL to discharge some TCC's. For details talk to Mark Snyder.

- We are developing a library in Haskell for processing Rosetta specifications. A current focus is the modularity and re-use of distinct processing elements, such as type-checking, partial evaluation, and reasoning assistants. Mutually defined elements that are more convenient to consider as distinct interact via a reactive monadic computation, so the two elements' code can be managed as separate packages. Also, our principal specification representation uses functors and type-level fixed points to achieve extensibility and generic programming. The goal of the library is to provide a tight and graduated interface to the basic processing elements, so that the users may incorporate the most appropriate basic elements when implementing their own, more domain-specific Rosetta processors. For details talk to Nick Frisby.

- CSDL is developing Oread (http://haskell.org/communities/11-2008/html/report.html#sect6.9.4), a language utilizing monadic concepts capturing message-passing concurrency, to explore the application of functional languages to hardware/software system design and implementation. The Oread toolset, implemented in Haskell, is used to compile the language to either embedded processor cores or FPGA hardware technology. The compiler is capable of emitting LLVM code, which can then be compiled to the Microblaze soft processor, or VHDL and Verilog, for direct implementation on Xilinx FPGA devices. For details talk to Garrin Kimmell.

We also lose Garrin Kimmell in June, when he moves to Iowa.

Further reading

- The Functional Programming Group (with a new website) http://www.ittc.ku.edu/csdl/fpg.
- CSDL website: https://wiki.ittc.ku.edu/csdl/Main_Page

8.9 Ghent Functional Programming Group

Report by:	Jeroen Janssen
Participants:	Bart Coppens, Jasper Van der Jeugt
Status:	starting up

The Ghent Functional Programming Group is a new user group aiming to bring together programmers, academics, and others interested in functional programming located in the area of Ghent, Belgium. Our goal is to have regular meetings with talks on functional programming, organize functional programming related events such as hackathons, and to promote functional programming in Ghent by giving after-hours tutorials. We had our first meeting on April 1, 2010 and welcomed almost 30 people. The first meeting consisted of a number of talks:

1. Jeroen Janssen — "Welcome and short introduction to Functional Programming"

2. Jasper Van der Jeugt — "BlazeHtml: a blazingly fast html generator in Haskell"

3. Tom Schrijvers — "Functional Pearl: The Monad Zipper"

4. Romain Slootmaekers — "Functional Programming at Amplidata: a tentative experience report"

The second meeting was on May 13, 2010. The program was as follows:

1. Atze Dijkstra — "The Utrecht Haskell Compiler"

2. Jean-Christophe Mincke — “An Introduction To Monads”

3. Drinks at a local bar

We are currently in the process of planning our third meeting, which will take place around the end of June. For more information you can follow us on twitter (@ghentfpg), via google groups (http://groups.google.com/group/ghent-fpg), or by visiting us at irc.freenode.net in channel #ghentfpg. We hope to be able to greet you at one of our next meetings.

Further reading

http://groups.google.com/group/ghent-fpg