Haskell Communities and Activities Report

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

Thirteenth Edition – December 22, 2007

Andres Löh (ed.)

Lloyd Allison	alpheccar	Tiago Miguel Laureano Alves
Krasimir Angelov	Apfelmus	Carlos Areces
Sengan Baring-Gould	Alistair Bayley	Clifford Beshers
Chris Brown	Bjorn Buckwalter	Andrew Butterfield
Manuel Chakravarty	Olaf Chitil	Duncan Coutts
Nils Anders Danielsson	Robert Dockins	Frederik Eaton
Keith Fahlgren	Jeroen Fokker	Simon Frankau
Leif Frenzel	Richard A. Frost	Clemens Fruhwirth
Andy Gill	George Giorgidze	Daniel Gorin
Martin Grabmüller	Murray Gross	Jurriaan Hage
Kevin Hammond	Bastiaan Heeren	Christopher Lane Hinson
Guillaume Hoffmann	Paul Hudak	Liyang Hu
Graham Hutton	Wolfgang Jeltsch	Antti-Juhani Kaijanaho
Oleg Kiselyov	Dirk Kleeblatt	Lennart Kolmodin
Slawomir Kolodynski	Eric Kow	Huiqing Li
Andres Löh	Rita Loogen	Salvador Lucas
Ian Lynagh	Ketil Malde	Christian Maeder
Simon Marlow	Steffen Mazanek	Conor McBride
Neil Mitchell	Andy Adams-Moran	Dino Morelli
Yann Morvan	Matthew Naylor	Rishiyur Nikhil
Stefan O'Rear	Simon Peyton-Jones	Dan Popa
Claus Reinke	David Roundy	Alberto Ruiz
David Sabel	Uwe Schmidt	Ganesh Sittampalam
Anthony Sloane	Dominic Steinitz	Don Stewart
Jennifer Streb	Martin Sulzmann	Doaitse Swierstra
Wouter Swierstra	Hans van Thiel	Henning Thielemann
Peter Thiemann	Simon Thompson	Phil Trinder
Andrea Vezzosi	Miguel Vilaca	Joost Visser
Janis Voigtländer	Edsko de Vries	Malcolm Wallace
Mark Wassell	Ashley Yakeley	Bulat Ziganshin

Preface

This is the 13th edition of the Haskell Communities and Activities Report, and it arrives just in time for the break between the years – if you are bored by all the free time you might suddenly have, why not sit down and study what other Haskellers have been up to during the past six months?

As always, entries that are completely new (or have been revived after disappearing temporarily from the edition) are formatted using a blue background. Updated entries have a header with a blue background. In the most cases, I have dropped entries that have not been changed for a year or longer.

Many thanks to all the contributors. A special "thank you" to the many contributors that have attempted to reduce my workload this year by sending their entries in the preferred LaTeX style – more than ever before: This has made the experience of assembling the report an even greater pleasure!

An interesting idea can be found in the Ansemond LLC entry (→ 7.1.1), where a screenshot is included. I would like the report to become more colourful and have more pictures. So, for previous editions, if you would like to include a screenshot along with your Haskell-related tool or application, please send it along with your entry.

Many Haskell projects exist now, and most of them seem to be looking for developers. If you are an enthusiastic Haskell programmer, please consider supporting one of the existing projects by offering your help, and please don't forget some of the "older", yet still very successful projects such as Darcs (→ 6.13) and Cabal (→ 4.1.1) over the continuous stream of new project and software announcements.

Despite the fun it has been, my time as editor of the Haskell Communities and Activities Report is coming to an end. *I am therefore looking for a new editor who would like to take over and continue the report, possibly adapting it to her or his own vision. Please contact me if you are interested.* A separate announcement will follow.

If a new editor can be found, we might prepare the next edition together, probably around May, so watch the mailing lists around this time for announcements – we continue to depend and rely on your contributions!

Feedback is of course very welcome ⟨hcar@haskell.org⟩. Enjoy the Report!

Andres Löh, Universiteit Utrecht, The Netherlands

Contents

1 General

1.1 HaskellWiki and haskell.org

Report by:	Ashley Yakeley

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

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

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

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

Further reading

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

1.2 `#haskell`

Report by:	Don Stewart

The `#haskell` IRC channel is a real-time text chat where anyone can join to discuss Haskell. The channel has continued to grow in the last 6 months, now averaging around 390 users, with a record 436 users. It is one of the largest channels on freenode. The irc channel is home to hpaste and lambdabot (→ 6.14), two useful Haskell bots. Point your IRC client to irc.freenode.net and join the `#haskell` conversation!

For non-English conversations about Haskell there is now:

- `#haskell.de` – German speakers
- `#haskell.dut` – Dutch speakers
- `#haskell.es` – Spanish speakers
- `#haskell.fi` – Finnish speakers
- `#haskell.fr` – French speakers
- `#haskell.hr` – Croatian speakers
- `#haskell.it` – Italian speakers
- `#haskell.jp` – Japenese speakers
- `#haskell.no` – Norwegian speakers
- `#haskell_ru` – Russian speakers
- `#haskell.se` – Swedish speakers

Related Haskell channels are now emerging, including:

- `#haskell-overflow` – Overflow conversations
- `#haskell-blah` – Haskell people talking about anything except Haskell itself
- `#gentoo-haskell` – Gentoo/Linux specific Haskell conversations (→ 7.4.3)
- `#haskell-books` – Authors organizing the collaborative writing of the Haskell +wikibook
- `#darcs` – Darcs revision control channel (written in Haskell) (→ 6.13)
- `#ghc` – GHC developer discussion (→ 2.1)
- `#happs` – HAppS Haskell Application Server channel
- `#xmonad` – Xmonad a tiling window manager written in Haskell (→ 6.3)

Further reading

More details at the `#haskell` home page: http://haskell.org/haskellwiki/IRC_channel

1.3 Planet Haskell

Report by:	Antti-Juhani Kaijanaho
Status:	(slightly) updated

Planet Haskell is an aggregator of Haskell people's blogs and other Haskell-related news sites. As of mid-November 2007 content from 78 blogs and other sites is being republished in a common format.

A common misunderstanding about Planet Haskell is that it republishes only Haskell content. That is not its mission. A Planet shows what is happening in the community, what people are thinking about or doing. Thus Planets tend to contain a fair bit of "off-topic" material. Think of it as a feature, not a bug.

For information on how to get added to Planet, please read http://planet.haskell.org/policy.html.

Further reading

http://planet.haskell.org/

1.3.1 Haskell Weekly News

Report by: Don Stewart

The Haskell Weekly News (HWN) is a weekly newsletter covering developments in Haskell. Content includes announcements of new projects, jobs, discussions from the various Haskell communities, notable project commit messages, Haskell in the blogspace, and more. The Haskell Weekly News also publishes latest releases uploaded to Hackage.

It is published in html form on The Haskell Sequence, via mail on the Haskell mailing list, on Planet Haskell (→ 1.3), and via RSS. Headlines are published on haskell.org (→ 1.1).

Further reading

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

1.4 The Monad.Reader

Report by: Wouter Swierstra

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

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

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

Since the last HCAR there have been two new issues, including a special issue about this year's Summer of Code. I'm always interested in new submissions, whether you're an established researcher or fledgling Haskell programmer. Check out the Monad.Reader homepage for all the information to you need to start writing your article.

Further reading

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

1.5 Books and tutorials

1.5.1 New textbook – Programming in Haskell

Report by: Graham Hutton

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

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

Publication details:

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

In-depth review:

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

Further information:

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

1.5.2 Haskell Wikibook

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

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

Since the last report, the wikibook has been progressing slowly but steadily. A chapter about Applicative Functors has been added, the module about Monads is being rewritten and comprehensive material about graph reduction and lazy evaluation is beginning to emerge. Thanks to the authors and to the many contributors that spot mistakes and ask those questions we'd never thought of!

Further reading

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

1.5.3 Gtk2Hs tutorial

Report by: Hans van Thiel

Part of the original GTK+2.0 tutorial by Tony Gail and Ian Main has been adapted to Gtk2Hs (→4.8.3), which is the Haskell binding to the GTK GUI library.

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

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

Available, at the time of writing (November 2007):

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

The completed tutorial will consist of ten or more chapters and will also build on "Programming with gtkmm" by Murray Cumming et al. and the Inti (Integrated Foundation Classes) tutorial by the Inti team. Completion is expected in 2Q 2008.

The Glade tutorial, an introduction to visual Gtk2Hs programming, has been updated to Glade 3 by to Alex Tarkovsky. It is available on: http://haskell.org/gtk2hs/docs/tutorial/glade/

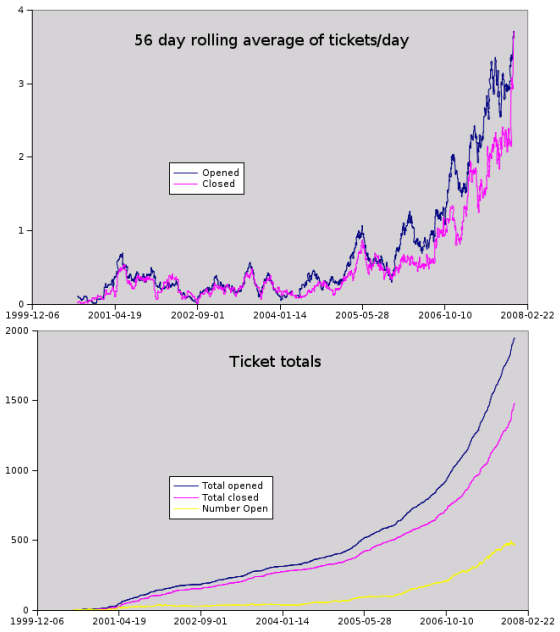
2 Implementations

2.1 The Glasgow Haskell Compiler

Report by: Simon Peyton-Jones, Simon Marlow, Norman Ramsey, Manuel Chakravarty, Ian Lynagh and many others

Lots has happened on the GHC front over the last few months. We released GHC 6.8.1 on 3 November 2007. GHC now has so many users, and such a large feature "surface area", that simply getting to the point where we can make a release is becoming quite a challenge. Indeed, a good deal of our effort in the last six months has been in the form of consolidation: fixing bugs and solidifying what we have.

These graphs show "tickets" which include bugs, feature requests, and tasks. Of the "open tickets", about half are bugs. Notice the big spike in "closed tickets" just before the 6.8.1 release!



The major new features of 6.8.1 were described in the last issue of the Haskell Communities Newsletter, so we won't repeat them here. Instead, here are some of the highlights of what we are working on now.

Syntactic and front-end enhancements

Several people have developed syntactic innovations, which are (or will shortly be) in the HEAD:

Three improvements to records

- *Wild-card patterns for records.* If you have

```
data T = MkT {x,y::Int, z::Bool}
```

then you can say

```
f :: T -> Int
f (MkT {..}) = x+y

g :: Int -> Int -> T
g x y = MkT {..}
      where
        z = x>y
```

The "`..`" in a pattern brings into scope all the fields of the record; while in a record construction it uses variables with those names to initialise the record fields. Here's the user manual entry: http://www.haskell.org/ghc/dist/current/docs/users_guide/syntax-extns.html#record-wildcards.

- *Record puns* is a slightly less abbreviated approach. You can write `f` like this:

```
f (MkT {x,y}) = x+y
```

whereas Haskell 98 requires you to write `x=x,y=y` in the pattern. Similarly in record construction.

- *Record field disambiguation* is useful when there are several types in scope, all with the same field name. For example, suppose another data type `S` had an `x` field. Then if you write

```
h (MkT {x=p,y=q}) = ...
```

there is no doubt which `x` you mean, but Haskell 98 will complain that there are two `x`s in scope. Record field disambiguation just uses the constructor to decide which `x` you must mean.

None of these changes tackle the deeper issue of whether or not Haskell's current approach to records is the Right Way; rather the changes just make the current approach work a bit better. Furthermore, they are all somewhat controversial, because they make it harder to see where something comes into scope. Let's see what you think!

View patterns

View patterns are implemented, by Dan Licata. Here's a simple example:

```
polar :: Complex -> (Float, Float)
polar = ...

f :: Complex -> Bool
f (polar -> (r,theta)) = r <= 1
```

Here `polar` is an ordinary function, used to transform the Complex to polar form. The view pattern is the argument pattern for `f`. Many details here: http://hackage.haskell.org/trac/ghc/wiki/ViewPatterns.

Generalised list comprehensions

Generalised list comprehensions (see Comprehensive comprehensions: comprehensions with "Order by" and "Group by", Phil Wadler and Simon Peyton Jones, Haskell Workshop 2007) have been implemented by Max Bolinbroke. Example:

```
[ (the dept, sum salary)
| (name, dept, salary) <- employees
, then sortWith by salary
, then takeWhile by salary < 50
, then take 5 ]
```

More details here: http://hackage.haskell.org/trac/ghc/wiki/SQLLikeComprehensions.

Quasi-quoting

We are keen to get Geoff Mainland's quasi-quoting mechanism into GHC (see "Why It's Nice to be Quoted: Quasiquoting for Haskell", Geoffrey Mainland. Haskell Workshop 2007). Geoff is working on polishing it up.

Type system stuff

The big innovation in GHC's type system has been the gradual introduction of indexed type families in the surface syntax, and of type equalities in the internal machinery.

Indexed data families (called "associated data types" when declared in type classes) are fairly simple, and they work fine in GHC 6.8.1. Indexed type families (aka "associated type synonyms") are a different kettle of fish, especially when combined with the ability to mention type equalities in overloaded types, thus:

```
f :: forall a b. (a ~ [b]) => ...
```

Tom Schrijvers spent three months at Cambridge, working on the theory and implementation of a type inference algorithm. As a result we have a partially-working implementation, and we understand the problem much better, but there is still much to do, both on the theoretical and practical front. It's trickier than we thought! We have a short paper Towards open type functions for Haskell which describes some of the issues, and a wiki page (http://hackage.haskell.org/trac/ghc/wiki/TypeFunctions) that we keep up to date; it has a link to details of implementation status. This is all joint work with Martin Sulzmann, Manuel Chakravarty, and Tom Schrijvers.

Parallel GC

Since 6.6 GHC has had support for running parallel Haskell on a multi-processor out of the box. However, the main drawback has been that the garbage collector is still single-threaded and stop-the-world. Since GC can commonly account for 30% of runtime (depending on the GC settings), this can seriously put a crimp in your parallel speedup.

Roshan James did an internship at MSR in 2006 during which he and Simon M worked on parallelising the major collections in GHC's generational garbage collector. We had a working algorithm, but didn't observe much speedup on a multi-processor. Since then, Simon rewrote the implementation and spent a large amount of time with various profiling tools, which uncovered some cache-unfriendly behaviour. We are now seeing some speedup, but there is more tweaking and measuring still to be done.

This parallel GC is likely to be in GHC 6.10. Note that parallel GC is independent of whether the Haskell program itself is parallel – so even single-threaded Haskell programs (e.g. GHC itself) should benefit from it.

The other side of the coin is to parallelise the *minor* collections. These are normally too small and quick to apply the full-scale parallel GC to, and yet the whole system still has to stop to perform a minor GC. The solution is almost certainly to allow each CPU to GC its own nursery independently. There is existing research describing how to do this, and we plan to try applying it in context of GHC.

Data parallel Haskell

After many months of designing, re-designing, and finally implementing a vectorisation pass operating on GHC's Core intermediate language, we finally have a complete path from nested data parallel array programs to the low-level, multi-threaded array library in package ndp. We are very excited about having reached this milestone, but the path is currently very thin, complete unoptimised, and requires a special Prelude mockup. More work is required before vectorisation is ready for end-users, but now that the core infrastructure is in place, we expect more rapid progress on user-visible features.

Besides working on optimisations and completing the backend library, we still need to implement Partial Vectorisation of Haskell Programs (http://www.cse.unsw.

edu.au/~chak/papers/CLPK07.html) and the treatment of unboxed types, which is crucial to vectorise the standard Prelude. Most of the code was written by Roman Leshchinskiy.

Back end stuff

GHC's back end code generator has long been known to generate poor code, particularly for tight loops of the kind that are cropping up more and more in highly optimised Haskell code. So in typical GHC style, rather than patch the immediate problem, we're redesigning the entire back end.

What we want to do:

- Split the STG-to-C-- code generator (codeGen) into two: one pass generating C-- with functions and calls, and a second pass ("CPS") to manifest the stack and calling/return conventions.
- Redesign the calling and return conventions, so that we can use more registers for parameter passing (this will entail decommissioning the via-C code generator, but the native code generator will outperform it).
- Give the back end more opportunity to do low-level transformation and optimisation, e.g. by exposing loops at the C-- level.
- Implement more optimisations over C--.
- Plug in a better register allocator.

What we've done so far:

- Michael Adams came for an internship and built a CPS converter for GHC's internal C-- data type.
- He had barely left when Norman Ramsey arrived for a short sabbatical. Based on his experience of building back ends for the Quick C-- compiler, he worked on a new zipper-based data structure to represent C-- code, and a sophisticated dataflow framework so that you can write new dataflow analyses in 30 mins.
- Ben Lippmeir spent his internship building a graph-colouring, coalescing register allocator for GHC's native code generator.

As a result, we now have *lots* of new code. Some of it is working; much of it is as yet un-integrated and un-tested. However, once we have it all glued back together, GHC will become a place where you can do Real Work on low-level optimisations, and code generation. Indeed John Dias (one of Norman's graduate students) will spend six months here in 2008 to do work on code generation.

In short, GHC's back end, which has long been a poor relation, is getting a lot of very sophisticated attention. Expect good things.

Libraries

GHC ships with a big bunch of libraries. That is good for users, but it has two bad consequences, both of which are getting worse with time. First, it make it much harder to get a release together, because we have to test more and more libraries too. Second, it's harder (or perhaps impossible) to upgrade the libraries independently from GHC. There's a meta-issue too: it forces us into a gate-keeper role in which a library gets a big boost by being in the "blessed set" shipped with GHC.

Increasingly, therefore, we are trying to decouple GHC from big libraries. We ship GHC with a set of "boot" libraries, without which GHC will not function at all, and "extra" libraries, which just happen to come with some binary distributions of GHC, and which can be upgraded separately at any time. To further that end, we've split the "base" package into a bunch of smaller packages, and expect to further split it up for GHC 6.10. This has led to lots of pain, because old programs that depended on 'base' now need to depend on other packages too; see upgrading packages (http://www.haskell.org/haskellwiki/Upgrading_packages) for details. But it's good pain, and matters should improve too as Cabal matures. We have been exploring possibilities for lessening the pain in 6.10: http://hackage.haskell.org/trac/ghc/wiki/PackageCompatibility. We have also devised a package versioning policy which will help future library upgrades: http://www.haskell.org/haskellwiki/Package_versioning_policy.

2.2 yhc

Report by: Neil Mitchell

The York Haskell Compiler (yhc) is a fork of the nhc98 compiler, with goals such as increased portability, platform independent bytecode, integrated Hat support and generally being a cleaner code base to work with. Yhc now compiles and runs almost all Haskell 98 programs, has basic FFI support – the main thing missing is haskell.org base libraries, which is being worked on.

Since that last HCAR we have continued to improve our Yhc.Core library, making use of it in a number of projects (optimisers, analysis tools) to be made available shortly. The Javascript back end has undergone lots of improvements with new libraries for writing dynamic web pages.

Further reading

- Homepage:
  http://www.haskell.org/haskellwiki/Yhc
- Darcs repository:
  http://darcs.haskell.org/yhc

2.3 The Helium compiler

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

Helium is a compiler that supports only a subset of Haskell (e.g., n+k patterns are missing). Moreover, 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 Helium compiler is still available for Download from http://www.cs.uu.nl/helium/.

At this moment, we are working on making version 1.7 available. Internally little will change except that the interface to Helium will be generalized so that multiple versions of Helium can side by side (motivated by the development of Neon) and that the logging facility can be more easily used outside our own environment. The loggings obtained in classes outside our university may help to improve the external validity of studies performed by using Neon (→ 4.2.2).

3 Language

3.1 Variations of Haskell

3.1.1 Liskell

Report by:	Clemens Fruhwirth
Status:	experimental

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

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

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

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

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

Further reading

http://liskell.org

3.1.2 Haskell on handheld devices

Report by:	Anthony Sloane
Participants:	Michael Olney
Status:	unreleased

The project at Macquarie University (→ 7.3.6) to run Haskell on handheld devices based on Palm OS has a running implementation for small tests but, like most ports of languages to Palm OS, we are dealing with memory allocation issues. Also, other higher priority projects have now intervened so this project is going into the background for a while.

3.2 Non-sequential Programming

3.2.1 GpH – Glasgow Parallel Haskell

Report by:	Phil Trinder
Participants:	Phil Trinder, Abyd Al Zain, Greg Michaelson, Kevin Hammond, Jost Berthold, Murray Gross

Status

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

System Evaluation and Enhancement

- A major revision of the parallel runtime environment for GHC 6.8 is currently under development. The GpH and Eden parallel Haskells share much of the implementation technology and both are being used for parallel language research and in the SCIEnce project (see below).

- We have developed an adaptive runtime environment (GRID-GUM) for GpH on computational grids. GRID-GUM incorporates new load management mechanisms that cheaply and effectively combine static and dynamic information to adapt to the heterogeneous and high-latency environment of a multi-cluster computational grid. We have made comparative measures of GRID-GUM's performance on high/low latency grids and heterogeneous/homogeneous grids using clusters located in Edinburgh, Munich and Galashiels. Results are published in:

Al Zain A. *Implementing High-Level Parallelism on Computational Grids*, PhD Thesis, Heriot-Watt University, 2006.

Al Zain A. Trinder P.W. Loidl H.W. Michaelson G.J. *Evaluating a High-Level Parallel Language (GpH) for Computational Grids*, IEEE Transactions on Parallel and Distributed Systems (February 2008).

- SMP-GHC, an implementation of GpH for multi-core machines has been developed by Tim Harris, Satnam Singh, Simon Marlow and Simon Peyton Jones.
- We are teaching parallelism to undergraduates using GpH at Heriot-Watt and Phillips Universitat Marburg.

GpH Applications

- As part of the SCIEnce EU FP6 I3 project (026133) (→ 7.3.10) (April 2006 - April 2011) we use GpH and Eden to provide access to computational grids from Computer Algebra(CA) systems, including GAP, Maple MuPad and KANT. We have implemented an interface, GCA, which orchestrates computational algebra components into a high-performance parallel application. GCA is capable of exploiting a variety of modern parallel/multicore architectures without any change to the underlying code. GCA is also capable of orchestrating heterogeneous computations across a high-performance computational Grid.

Implementations

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

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

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

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

Further reading

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

Contact

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

3.2.2 Eden

Report by: Rita Loogen

Description

Eden has been jointly developed by two groups at Philipps Universität Marburg, Germany and Universidad Complutense de Madrid, Spain. The project has been ongoing since 1996. Currently, the team consists of the following people:

in Madrid: Ricardo Peña, Yolanda Ortega-Mallén, Mercedes Hidalgo, Fernando Rubio, Clara Segura, Alberto Verdejo

in Marburg: Rita Loogen, Jost Berthold, Steffen Priebe, Mischa Dieterle

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

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

Survey and standard reference

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

Implementation

A major revision of the parallel Eden runtime environment for GHC 6.8.1 is available on request. Support

for Glasgow parallel Haskell (GpH) is currently being added to this version of the runtime environment. It is planned for the future to maintain a common parallel runtime environment for Eden, GpH and other parallel Haskells.

Recent and Forthcoming Publications

- Mischa Dieterle: *Parallel functional implementation of Master-Worker-Skeletons*, Diploma Thesis, Philipps-University Marburg, October 2007 (in German).
- Jost Berthold, Mischa Dieterle, Rita Loogen: *Functional Implementation of a Distributed Work Pool Skeleton*, submitted.
- Jost Berthold, Mischa Dieterle, Rita Loogen, Steffen Priebe: *Hierarchical Master-Worker Skeletons*, Practical Aspects of Declarative Languages (PADL) 08, San Francisco, USA, January 2008, LNCS, Springer, to appear.
- Jost Berthold, Abyd Al-Zain, and Hans-Wolfgang Loidl: *Adaptive High-Level Scheduling in a Generic Parallel Runtime Environment*, Practical Aspects of Declarative Languages (PADL) 08, San Francisco, USA, January 2008, LNCS, Springer, to appear.
- Jost Berthold and Rita Loogen: *Visualising Parallel Functional Program Runs - Case Studies with the Eden Trace Viewer*, Parallel Computing: Architectures, Algorithms and Applications, Proceedings of the International Conference ParCo 2007, NIC-Series, to appear.
- Mercedes Hidalgo-Herrero and Yolanda Ortega-Mallén: *To be or not to be ... lazy*, In Draft Proceedings of 19th Intl. Symposium on the Implementation of Functional Languages (IFL 2007), University of Kent, Canterbury (UK) 2007.
- A. de la Encina, L. Llana, F. Rubio, M. Hidalgo-Herrero: *Observing Intermediate Structures in a Parallel Lazy Functional Language*, 9th International ACM-SIGPLAN Symposium on Principles and Practice of Declarative Programming, PPDP'07, ACM Press 2007, pages 111-120.
- Mercedes Hidalgo-Herrero, Alberto Verdejo, Yolanda Ortega-Mallén: *Using Maude and its strategies for defining a framework for analyzing Eden semantics*, WRS 06 (6th International Workshop on Reduction Strategies in Rewriting and Programming), Electronic Notes in Theoretical Computer Science, Volume 174, Issue 10, Pages 119-137 (July 2007).
- A. de la Encina, I. Rodríguez, F. Rubio: *Testing Speculative Work in a Lazy / Eager Parallel Functional Language.*, LCPC'05, LNCS 4339, Springer 2007.

Further reading

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

3.3 Type System/Program Analysis

3.3.1 Free Theorems for Haskell

Report by:	Janis Voigtländer
Participants:	Sascha Böhme and Florian Stenger

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. For recent application papers, see "Proving Correctness via Free Theorems: The Case of the destroy/build-Rule" (PEPM'08) and "Much Ado about Two: A Pearl on Parallel Prefix Computation" (POPL'08).

Also on the practical side, Sascha Böhme implemented a library and tool for generating free theorems from Haskell types. Downloadable source and a web interface are accessible at http://linux.tcs.inf.tu-dresden.de/~voigt/ft. Features include:

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

While the web interface is restricted to algebraic data types, type synonyms, and type classes from Haskell standard libraries, a shell-based application contained in the source package also enables the user to declare their own algebraic data types and so on, and then to derive free theorems from types involving those.

Further reading

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

3.3.2 Agda

Report by:	Nils Anders Danielsson
Status:	Actively developed by a number of people

Do you crave for highly expressive types, but do not want to resort to type-class hackery? Then Agda might

provide a view of what the future has in store for you.

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

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

Further reading

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

3.3.3 Epigram

Report by:	Conor McBride

Epigram is a prototype dependently typed functional programming language, equipped with an interactive editing and typechecking environment. High-level Epigram source code elaborates into a dependent type theory based on Zhaohui Luo's UTT. The definition of Epigram, together with its elaboration rules, may be found in 'The view from the left' by Conor McBride and James McKinna (JFP 14 (1)).

A new version, Epigram 2, based on Observational Type Theory (see 'Observational Equality, Now!' by Thorsten Altenkirch, Conor McBride, and Wouter Swierstra) is in preparation.

Motivation

Simply typed languages have the property that any subexpression of a well typed program may be replaced by another of the same type. Such type systems may guarantee that your program won't crash your computer, but the simple fact that True and False are always interchangeable inhibits the expression of stronger guarantees. Epigram is an experiment in freedom from this compulsory ignorance.

Specifically, Epigram is designed to support programming with inductive datatype families indexed by data. Examples include matrices indexed by their dimensions, expressions indexed by their types, search trees indexed by their bounds. In many ways, these datatype families are the progenitors of Haskell's GADTs, but indexing by data provides both a conceptual simplification – the dimensions of a matrix are *numbers* – and a new way to allow data to stand as *evidence* for the properties of other data. It is no good representing sorted lists if comparison does not produce evidence of ordering. It is no good writing a type-safe interpreter if one's typechecking algorithm cannot produce well-typed terms.

Programming with evidence lies at the heart of Epigram's design. Epigram generalises constructor pattern matching by allowing types resembling induction principles to express as how the inspection of data may affect both the flow of control at run time and the text and type of the program in the editor. Epigram extracts patterns from induction principles and induction principles from inductive datatype families.

History

James McKinna and Conor McBride designed Epigram in 2001, whilst based at Durham, working with Zhaohui Luo and Paul Callaghan. McBride's prototype implementation of the language, 'Epigram 1' emerged in 2004: it is implemented in Haskell, interfacing with the xemacs editor. This implementation effort involved inventing a number of new programming techniques which have found their way into the Haskell community at large: central components of Control.Applicative and Data.Traversable started life in the source code for Epigram.

Following the Durham diaspora, James McKinna and Edwin Brady went to St. Andrews, where they continued their work on phase analysis and efficient compilation of dependently typed programs. More recently, with Kevin Hammond, they have been studying applications of dependent types to resource-aware computation in general, and network protocols in particular.

Meanwhile, Conor McBride went to Nottingham to work with Thorsten Altenkirch. They set about redesigning Epigram's underlying type theory, radically changing its treatment of logical propositions in general, and *equality* in particular, making significant progress on problems which have beset dependent type theories for decades.

The Nottingham duo grew into a strong team of enthusiastic researchers. Peter Morris successfully completed a PhD on generic programming in Epigram and is now a research assistant: his work has led to the redesign of Epigram's datatype language. Nicolas Oury joined from Paris as a postdoctoral research fellow, and is now deeply involved in all aspects of design and implementation. PhD students James Chapman and Wouter Swierstra are working on Epigram-related topics, studying formalized metatheory and effectful programming, respectively. Meanwhile, Nottingham research on *containers*, involving Neil Ghani, Peter Hancock and Rawle Prince, together with the Epigram team, continues to inform design choices as the language evolves.

Epigram 1 was used successfully by Thorsten Altenkirch, Conor McBride, and Peter Hancock in an undergraduate course on Computer Aided Formal Reasoning http://www.e-pig.org/darcs/g5bcfr/. It has also been used in a number of graduate-level courses.

James McKinna is now at Radboud University, Nijmegen; Edwin Brady is still at St. Andrews; Thorsten Altenkirch, Peter Morris, Nicolas Oury, James Chapman and Wouter Swierstra are still in Nottingham; Conor McBride has left academia. All are still contributing to the Epigram project.

Current Status

Epigram 2 is based on a radical redesign of our underlying type theory. The main novelties are

- a *bidirectional* approach to typechecking, separating syntactically the terms whose types are inferred from those for which types are pushed in—with stronger guarantees of prior type information, we can reduce clutter in terms and support greater overloading;
- explicit separation of *propositions* and *sets*, ensuring that proofs never influence control-flow and can be erased at run-time;
- a *type-directed* approach to propositional equality, comparing functions extensionally, records componentwise, data by construction, and proofs trivially—we shall soon support equality for codata by bisimulation and for quotients by whatever you want;
- three *closed universes* of data structures, finite enumerations, record types, and inductive datatypes, each with its datatype of type descriptions—this supports generic programming over all of Epigram 2's data structures and removes the need for any means of 'making new stuff' other than definition.

Nicolas Oury, Peter Morris, and Conor McBride have implemented this theory, together with a system supporting interactive construction (and destruction) within it. This the engine which will drive Epigram 2: we plan to equip it with human-accessible controls and release it for the benefit of the curious, shortly. With this in place, we shall reconstruct the Epigram source language and its elaboration mechanism: constructs in source become constructions in the core.

There is still a great deal of work to do. We need to incorporate the work from Edwin Brady and James McKinna on type erasure and efficient compilation; we need to bring out and exploit the container structure of data; we need to support programming with effects (including non-termination); we need a declarative proof language, as well as a functional programming language.

The Epigram project relies on Haskell, its libraries, and tools such as alex (→ 5.2.1), happy (→ 5.2.2), bnfc, cabal (→ 4.1.1), and darcs (→ 6.13). We have recently developed tools for assembling the modules corresponding to each component of the Epigram system from files corresponding to each feature of the Epigram language: this may prove useful to others, so we hope to clean them up and release them. Meanwhile, as Haskell itself edges ever closer to dependent types, the Epigram project has ever more to contribute, in exploration of the design space, in the development of implementation technique, and in experimentation with the pragmatics of programming with such power and precision.

Epigram source code and related research papers can be found on the web at http://www.e-pig.org and its community of experimental users communicate via the mailing list ⟨epigram@durham.ac.uk⟩. The current, rapidly evolving state of Epigram 2 can be found at http://www.e-pig.org/epilogue/.

3.3.4 Chameleon project

Report by:	Martin Sulzmann

Chameleon is a Haskell style language which integrates sophisticated reasoning capabilities into a programming language via its CHR programmable type system. Thus, we can program novel type system applications in terms of CHRs which previously required special-purpose systems.

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

3.3.5 XHaskell project

Report by:	Martin Sulzmann
Participants:	Kenny Zhuo Ming Lu and Martin Sulzmann

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

Latest developments

We have fully implemented the system which can be used in combination with the Glasgow Haskell Compiler. We have taken care to provide meaningful type error messages in case the static checking of programs fails. Our system also allows to defer some static checks until run-time.

We make use of GHC-as-a-library so that the XHaskell programmer can easily integrate her programs into existing applications and take advantage of

the many libraries available in GHC. We also provide a convenient interface to the HaXML parser.

3.3.6 HaskellJoin

Report by:	Martin Sulzmann
Participants:	Edmund S. L. Lam and Martin Sulzmann

HaskellJoin extends Haskell with Join-calculus style concurrency primitives. The novelty lies in the addition of guards and propagated join patterns. These additional features prove to be highly useful. See for details: http://taichi.ddns.comp.nus.edu.sg/taichiwiki/HaskellJoinRules.

Latest developments

We have implemented a prototype in STM Haskell. Experimental results show that we can achieve significant speed-ups on multi-core architectures (more cores = programs runs faster).

HaskellJoin subsumes in expressive power "ADOM: Agent Domain of Monads" which is no longer supported.

3.3.7 Uniqueness Typing

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

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

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

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

Further reading

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

3.4 Backend

3.4.1 The Reduceron

Report by:	Matthew Naylor
Participants:	Colin Runciman, Neil Mitchell
Status:	Experimental

The Reduceron is a prototype of a special-purpose graph reduction machine, built using an FPGA, featuring:

- parallel, dual-port, quad-word, stack, heap and combinator memories
- two-cycle n-ary application node unwinding (where $n \leq 8$)
- octo-instantiation (8 words per cycle) of supercombinator bodies
- parallel instantiation of combinator spine to heap and stack

The Reduceron is an extremely simple machine, containing just four instructions, and executes core Haskell almost directly. The translator from Yhc.Core to Reduceron bytecode and the FPGA machine are both implemented in Haskell, the latter using Lava. Other notable differences since the initial release of Reduceron are:

- Performs supercombinator (not SK) reduction, with data types encoded as functions, inspired by Jan Martin Jansen's SAPL interpreter
- Uses entirely on-chip memories on a Xilinx Virtex-II FPGA
- Has a garbage collector (in hardware)
- Includes a basic bytecode interpreter written in C, which is competitive with the nhc98 compiler on a small set of examples
- Includes Lava support for multi-output primitives and Xilinx block RAMs
- Includes three new Lava modules: CircLib.hs, a prelude of common circuits; CascadedRam.hs, for constructing RAMs of various widths and sizes; and RTL.hs, for writing register-transfer level descriptions.

The URL below links to the latest code, details and results of the Reduceron experiment.

Further reading

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

4 Libraries

4.1 Packaging and Distribution

4.1.1 Cabal and HackageDB

Report by: Duncan Coutts

Background

The Haskell Cabal is a Common Architecture for Building Applications and Libraries. It is an API distributed with GHC (→ 2.1), nhc98, and Hugs which allows a developer to easily build and distribute packages.

HackageDB (Haskell Package Database) is an online database of packages which can be interactively queried via the website and client-side software such as cabal-install. From HackageDB, an end-user can download and install Cabal packages.

Recent progress

The last year has seen HackageDB take off. It has grown from a handful of packages to over 300. It has also seen the release of a major new version of the Cabal library – the 1.2.x series – which is bundled with recent GHC versions. This release was a big step forward in terms of new features, fewer rough edges and improved internal design.

Growing pains

The rapid growth of the HackageDB collection has highlighted some problems. There is now a lot of choice in packages but relatively little information to help users decide which package they want or whether it is likely to build on their platform.

Another problem is having to manually download and build packages and their dependencies. Fortunately this problem has a solution in the form of the command line tool `cabal-install` which has become increasingly usable in the last few months. The plan is for `cabal-install` to be the primary command line interface to Cabal and HackageDB, replacing `runhaskell Setup.lhs` and other `cabal-*` wrappers you may have heard of. Everyone is encouraged to preview this bright new future by trying the latest development versions of the Cabal library and cabal-install tool.

Looking forward

There is a great deal to do. The Cabal library needs a proper dependency framework. There are many good ideas for technical and social solutions to the current problems with HackageDB. Unfortunately, for something that is now a vital piece of community infrastructure, there are relatively few people working on the solutions. We would like to encourage people to get involved, join the development mailing list, get the code and check the bug tracker for what needs to be done.

Even if you do not have time for hacking, you probably have a favourite Cabal bug or limitation. Do not just assume it is well known. Make sure it is properly described on the bug tracker and add yourself to the *cc* list so Cabal hackers can get some impression of priorities.

People

Cabal has seen contributions from 39 people in the three and a half years since Isaac Jones started the project. By simplistically counting patches we see that 90% of the code is by the top 8 contributors who have 50 or more patches each. 5% is by the next 5 most active contributors with 10 or more patches each. Contributions from a further 26 people make up the remaining 5%.

Further reading

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

4.2 General libraries

4.2.1 HPDF

Report by: alpheccar
Status: Continuous development

HPDF is an Haskell library allowing to generate PDF documents. HPDF is supporting several features of the PDF standard like outlines, multi-pages, annotations, actions, image embedding, shapes, patterns, text.

In addition to the standard PDF features, HPDF is providing some typesetting features built on top of the PDF core. With HPDF, it is possible to define complex styles for sentences and paragraphs. HPDF is implementing an optimum-fit line breaking algorithm a bit like the TeX one and HPDF is using the standard Liang hyphenation algorithm.

HPDF is at version 1.3. It is progressing continuously.

HPDF is available on Hackage (→ 4.1.1).

There are several missing features: the only supported fonts are the standard PDF ones. A next version should support TrueType and different character encodings. For support of Asian languages, I'll ask for help in the Haskell community.

I also plan to define an API easing the definition of complex layouts (slides, books). Currently the layout has to be coded by hand but it is already possible to build complex things.

The documentation is a bit weak and will have to be improved.

Further reading

http://www.alpheccar.org

4.2.2 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 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 samples of loggings, but it will allow programmers to play with it.

4.2.3 Test.IOSpec

Report by:	Wouter Swierstra
Status:	active development

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

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

The current release is described by a recent Haskell Workshop paper. The development version in the darcs repository, however, supports several exciting new features, including a modular way to combine specifications and a specification of STM. I have used Test.IOSpec to test and debug several substantial programs, such as a distributed Sudoku solver. If you use Test.IOSpec for anything useful at all, I'd love to hear from you.

Further reading

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

4.2.4 GSLHaskell

Report by:	Alberto Ruiz
Status:	active development

GSLHaskell is a simple library for linear algebra and numerical computation, internally implemented using GSL, BLAS and LAPACK.

A new version with important changes has been recently released. The internal code has been rewritten, based on an improved matrix representation. The interface is now simpler and more generic. It works on Linux, Windows and Mac OS X.

The library is available from HackageDB (→ 4.1.1) with the new name "hmatrix" (because only a small part of GSL is currently available, and matrix computations are based on LAPACK).

Most linear algebra functions mentioned in GNU-Octave's Quick Reference are already available both for real and complex matrices: eig, svd, chol, qr, hess, schur, inv, pinv, expm, norm, and det. There are also functions for numeric integration and differentiation, nonlinear minimization, polynomial root finding, and more than 200 GSL special functions. A brief manual is available at the URL below.

This library is used in the *easyVision* project (→ 6.21).

Further reading

http://alberrto.googlepages.com/gslhaskell

4.2.5 An Index Aware Linear Algebra Library

Report by:	Frederik Eaton
Status:	unstable; actively maintained

The index aware linear algebra library is a Haskell interface to a set of common vector and matrix operations. The interface exposes index types to the type system so that operand conformability can be statically guaranteed. For instance, an attempt to add or multiply two incompatibly sized matrices is a static error.

The library should still be considered alpha quality. A backend for sparse vector types is near completion, which allows low-overhead "views" of tensors as arbitrarily nested vectors. For instance, a matrix, which we represent as a tuple-indexed vector, could also be seen as a (rank 1) vector of (rank 1) vectors. These different

views usually produce different behaviours under common vector operations, thus increasing the expressive power of the interface.

Further reading

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

4.3 Parsing and Transforming

4.3.1 Graph Parser Combinators

Report by:	Steffen Mazanek
Status:	research prototype

A graph language can be described by a graph grammar in a manner similar to a string grammar known from the theory of formal languages. Unfortunately, graph parsing is known to be computationally expensive in general. There are even context-free graph languages the parsing of which is NP-complete.

Therefore we have developed the Haskell library *graph parser combinators*, a new approach to graph parsing inspired by the well-known string parser combinators. The basic idea is to define primitive graph parsers for elementary graph components and a set of combinators for the construction of more advanced graph parsers. Using graph parser combinators efficient special-purpose graph parsers can be composed conveniently in a for Haskell programmers familiar manner.

The following features are already implemented:

- a module `PolyStateSet` that is an extension of `PolyState` of the polyparse library that can deal with sets of tokens
- graph type declarations for several purposes
- graph parser combinators for important graph patterns
- parsers for several example graph languages
- for comparison a general-purpose parser for hyperedge replacement graph grammars

The library will soon be provided via hackage (→ 4.1.1).

4.3.2 uniplate

Report by:	Neil Mitchell

Uniplate is a boilerplate removal library, with similar goals to the original Scrap Your Boilerplate work. It requires fewer language extensions, and allows more succinct traversals with higher performance than SYB. A paper including many examples was presented at the Haskell Workshop 2007c̃refhaskell-workshop.

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

Further reading

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

4.3.3 InterpreterLib

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

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

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

Further reading

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

Contact

⟨nfrisby@ittc.ku.edu⟩

4.3.4 hscolour

Report by:	Malcolm Wallace
Status:	stable, maintained

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

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

Further reading

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

4.3.5 Utrecht Parsing Library and Attribute Grammar System

Report by:	Doaitse Swierstra and Jeroen Fokker
Status:	Released as cabal packages

The Utrecht attribute grammar system has been extended:

- the attribute flow analysis has been completely implemented by Joost Verhoog, and it is now possible to generate visit-function based evaluators, which are much faster and use less space. We assume that such functions are strict in all their arguments, and generate the appropriate `'seq'` calls to make the GHC aware of this. As a result also `case`'s are generated instead on `let`'s wherever possible.

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

- we are adding better support for higher order attribute grammars and forwarding rules

- Tthe error correcting strategies of the parser combinators are now being used as a base for providing automatic feedback in systems for training strategies (Johan Jeuring, Arthur van Leeuwen)

- a start has been made with providing Haddock information with the code of the parser combinators

- we plan to enhance the parser combinators with a second basic parsing engine, in order to support monadic uses of the combinators while keeping the error correcting capabilities

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

4.3.6 The X-SAIGA Project (was: Left-Recursive Parser Combinators)

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

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

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

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

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

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

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

4.4 System

4.4.1 hspread

Report by:	Andrea Vezzosi
Participants:	Andrea Vezzosi, Jeff Muller
Status:	active

hspread is a client library for the Spread toolkit. It is fully implemented in Haskell using the binary package (→ 4.7.1) for fast parsing of network packets. Its aim is to make easier to implement correct distributed applications by taking advantage of the guarantees granted by Spread, such as reliable and total ordered messages, and supports the most recent version of the protocol.

There is interest in further developing an higher level framework for Haskell distributed programming by extending the protocol if necessary.

Further reading

- Hackage: http://hackage.haskell.org/cgi-bin/hackage-scripts/package/hspread
- Developement version: darcs get http://happs.org/repo/hspread
- Spread homepage: http://www.spread.org

4.4.2 Harpy

Report by:	Martin Grabmüller and Dirk Kleeblatt
Status:	experimental

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

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

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

A second release is forthcoming, featuring improvements in the memory management, better floating point instruction support, and named labels that are shown in the disassembler output.

Further reading

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

4.4.3 hs-plugins

Report by:	Don Stewart
Status:	maintained

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

hs-plugins has been ported to GHC 6.6.

Further reading

- Source and documentation can be found at: http://www.cse.unsw.edu.au/~dons/hs-plugins/
- The source repository is available: `darcs get` http://www.cse.unsw.edu.au/~dons/code/hs-plugins/

4.4.4 The libpcap Binding

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

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

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

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

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

4.5 Databases and data storage

4.5.1 Takusen

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

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

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

Currently we fully support Oracle, Sqlite, and PostgreSQL, and ODBC support exists but is not fully tested.

Since the last report we have:

- added an ODBC backend
- improved the installation process so that we can build Haddock docs with Cabal

A new release to promote the ODBC code should be forthcoming; until then interested souls can get the latest from the darcs repo.

Future plans

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

Further reading

- darcs get http://darcs.haskell.org/takusen/
- browse docs:
  http://darcs.haskell.org/takusen/doc/html
  (see Database.Enumerator for Usage instructions and examples)

4.6 Data types and data structures

4.6.1 Data.Record

Report by:	Claus Reinke
Status:	library sketch

Extensible records, or the lack thereof, continue to be a popular subject of discussion. There is no lack of proposals, some implemented, some not, but there seems to be no obvious winner that would justify substantial implementation efforts, not to mention upsetting Haskell's already featureful type system.

Ever since seeing the Trex in Hugs development in Nottingham, I had been wondering about whether there were smaller aspects of extensible record systems that might be added as general features to the current type system, so that a record system could be defined on top of the extended system. The latter turned out to be almost possible with the then prevailing Hugs type system, but for commutative row constructors (label ordering should not matter) and negated type predicates (record 'has' label vs. record 'lacks' label).

Since then, extensions to the HList library (→ 4.6.7) have demonstrated that one can abuse GHC's type system implementation to get just enough expressiveness for defining a record system (an impressive feat), provided one is prepared to define a global ordering on record field labels. Separately, Daan Leijen's scoped labels proposal suggested that accepting the absence of negative predicates leads to a different, but not necessarily worse record system.

With all these ideas in the air, I found myself needing an extensible record system for a modular attribute grammar problem and was surprised to find an implementation of such a system within the limitations of GHC's type system – Data.Record was born! It was based on scoped labels, but went further in providing

record concatenation as well. I first posted the module Data.Record as an attachment to a Haskell' ticket on type sharing, and to the Haskell' list as an example of how code using only language extensions nominally supported in both GHC and Hugs would nevertheless only work in GHC, not in Hugs.

One of the recent revivals of the extensible records discussion made me dust off that old code and add some of the features requested for alternative systems. In particular, there is now support for unscoped operations (negative predicates, no duplicate labels) and for record label permutation. The former means that this code could grow into a library supporting all the major extensible record system styles, the latter means that record code can be label-order independent without needing a global ordering on labels (a prerequisite in most other type-class-based extensible record systems).

The code is not currently in a release state, being an insufficiently systematic collection of features from several systems, but usable, with examples, and if there was sufficient interest, I could try getting it more organised. Please let me know if you like what is there sufficiently to warrant such an effort.

Further reading

http://www.cs.kent.ac.uk/~cr3/toolbox/haskell/#records

4.6.2 Data.ByteString

Report by:	Don Stewart
Status:	active development

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

Data.ByteString is written in Haskell98 plus the foreign function interface and cpp. It has been tested successfully with GHC 6.4, 6.6, 6.8, Hugs 2005–2006, and the head version of nhc98.

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

Further reading

- Source and documentation can be found at http://www.cse.unsw.edu.au/~dons/fps.html
- The source repository is available:
  `darcs get`
  http://darcs.haskell.org/bytestring

4.6.3 stream-fusion (was: Data.List.Stream)

Report by:	Don Stewart
Status:	active development

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

This library is under active development, and we expect to port the ndp and bytestring libraries to use it.

Further reading

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

4.6.4 Edison

Report by:	Rob Dockins
Status:	stable, maintained

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

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

In theory, either component may be used independently of the other.

I took over maintenance of Edison about 18 months ago in order to update Edison to use the most current Haskell tools. The following major changes have been made since version 1.1, which was released in 1999.

- Typeclasses updated to use fundeps (by Andrew Bromage)
- Implementation of ternary search tries (by Andrew Bromage)

- Modules renamed to use the hierarchical module extension
- Documentation haddockized
- Source moved to a darcs repository
- Build system cabalized
- Unit tests integrated into a single driver program which exercises all the concrete implementations shipped with Edison
- Multiple additions to the APIs (mostly the associated collection API)

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

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

Further reading

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

4.6.5 dimensional

Report by:	Bjorn Buckwalter
Status:	active, mostly stable

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

Following a reorganization of the module hierarchy the core of dimensional is now mostly stable while additional units are being added on an as-needed basis. In addition to the SI system of units dimensional has experimental support for user-defined dimensions and a proof-of-concept implementation of the CGS system of units.

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

Further reading

http://dimensional.googlecode.com

4.6.6 Numeric prelude

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

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

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

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

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

◦ algebra, e.g. multi-variate polynomial (basics only),
◦ permutation group.

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

Using the revised system requires hiding some of the standard functions provided by Prelude, which is fortunately supported by GHC (→ 2.1). The library has basic Cabal support and a growing test-suite of QuickCheck tests for the implemented mathematical objects.

Future plans

Collect more Haskell code related to mathematics, e.g. for linear algebra. Study of alternative numeric type class proposals and common computer algebra systems. Ideally each data type resides in a separate module. However this leads to mutual recursive dependencies, which cannot be resolved if type classes are mutually recursive. We start to resolve this by fixing the types of some parameters of type class methods. E.g. power exponents become simply Integer instead of Integral, which has also the advantage of reduced type defaulting.

A still unsolved problem arises for residue classes, matrix computations, infinite precision numbers, fixed point numbers and others. It should be possible to assert statically that the arguments of a function are residue classes with respect to the same divisor, or that they are vectors of the same size. Possible ways out are encoding values in types or local type class instances. The latter one is still neither proposed nor implemented in any Haskell compiler. The modules are implemented in a way to keep all options open. That is, for each number type there is one module implementing the necessary operations which expect the context as a parameter. Then there are several modules which provide different interfaces through type class instances to these operations.

Further reading

http://darcs.haskell.org/numericprelude/

4.6.7 HList – a library for typed heterogeneous collections

Report by:	Oleg Kiselyov
Developers:	Oleg Kiselyov, Ralf Lämmel, Keean Schupke

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

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

The HList repository is available via Darcs (→ 6.13): http://darcs.haskell.org/HList

The library is being optimized and extended. Since the last report, we have added `ConsUnion.hs` to build homogeneous lists of heterogeneous components by constructing the union on-the-fly. We added Template Haskell code to eliminate the annoying boilerplate when defining record 'labels'. We optimized record projection, which should be especially noticeable for record narrowing. We added `equivR`, record equivalence modulo field order, with witnessing conversions. `ConsUnion.hs` checks for record types and treat the latter equivalent modulo the order of fields. This gives optimized, shallower unions.

Further reading

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

4.7 Data processing

4.7.1 binary

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

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

The 'binary' package provides efficient serialisation of Haskell values to and from lazy ByteStrings. ByteStrings constructed this way may then be written to disk, written to the network, or further processed

(e.g. stored in memory directly, or compressed in memory with zlib or bzlib).

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

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

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

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

The package is is available through Hackage (→ 4.1.1).

Further reading

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

4.7.2 binarydefer

Report by: Neil Mitchell

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

This library is at the heart of Hoogle 4 (→ 5.5.6), but has already found uses outside that application, including to do offline sorts etc.

Further reading

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

4.7.3 The Haskell Cryptographic Library

Report by: Dominic Steinitz

The current version is still 4.0.3.

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

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

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

Several people have posted more efficient versions of SHA-1 but not as patches. Given my limited time, I haven't been able to do anything with these.

This release contains:

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

Further reading

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

4.7.4 The Haskell ASN.1 Library

Report by: Dominic Steinitz

The current release is 0.0.11 which contains functions to handle ASN.1, X.509, PKCS#8 and PKCS#1.5.

This still has a dependency on NewBinary.

The current version handles the Basic Encoding Rules (BER). In addition, a significant amount of work has been undertaken on handling the Packed Encoding Rules (PER) using a GADT to represent the Abstract Syntax Tree (we'll probably move the BER to use the same AST at some point). You can download the current working version and try the unit and QuickCheck property tests for PER. These are not yet built by Cabal.

This release supports:

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

Further reading

http://haskell.org/asn1.

4.7.5 2LT: Two-Level Transformation

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

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

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

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

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

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

Recently, the 2LT project has been migrated to Google Code. New functionality is planned, such as elaboration of the front-ends and the creation of a web interface.

Further reading

Project URL: http://2lt.googlecode.com

- Alcino Cunha, José Nuno Oliveira, Joost Visser. *Type-safe Two-level Data Transformation*. Formal Methods 2006.
- Alcino Cunha, Joost Visser. Strongly Typed Rewriting For Coupled Software Transformation. RULE 2006.
- Pablo Berdaguer, Alcino Cunha, Hugo Pacheco, Joost Visser. *Coupled Schema Transformation and Data Conversion For XML and SQL*. PADL 2007.
- Alcino Cunha and Joost Visser. *Transformation of Structure-Shy Programs, Applied to XPath Queries and Strategic Functions*. PEPM 2007.
- Tiago L. Alves, Paulo Silva and Joost Visser. *Constraint-aware Schema Transformation*. Draft, 2007.

4.8 User interfaces

4.8.1 Shellac

Report by:	Rob Dockins
Status:	beta, maintained

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

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

Shellac is available from Hackage, as is the related Shellac-readline package.

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

will likely be releasing an officially "stable" version in the not-too-distant future. I anticipate few changes from the current version.

Further reading

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

4.8.2 Grapefruit – A declarative GUI and graphics library

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

Grapefruit is a library for creating graphical user interfaces and animated graphics in a declarative way.

Fundamental to Grapefruit is the notion of signal. A signal denotes either a time-varying value (the continuous case) or a sequence of values assigned to discrete points in time (the discrete case). Signals can be constructed in a purely functional manner.

User interfaces are described as systems of interconnected components which communicate via signals. To build such systems, the methods from the Arrow and ArrowLoop classes are used. For describing animated graphics, a special signal type exists.

Grapefruit also provides list signals. A list signal is a list-valued signal which can be updated incrementally and thus efficiently. In addition, a list signal associates an identity with each element so that moving an element within the list can be distinguished from removing the element and adding it again. List signals can be used to describe dynamic user interfaces, i.e., user interfaces with a changing set of components and changing order of components.

Grapefruit descriptions of user interfaces and animations always cover their complete lifetime. No explicit event handler registrations and no explicit recalculations of values are necessary. This is in line with the declarative nature of Haskell because it stresses the behavior of GUIs and animations instead of how this behavior is achieved. Internally though, Grapefruit is implemented efficiently using a common event dispatching and handling mechanism.

Grapefruit is currently based on Gtk2Hs (→ 4.8.3) and HOpenGL but implementations on top of other GUI and graphics libraries are possible. The aim is to provide alternative implementations based on different GUI toolkits so that a single application is able to integrate itself into multiple desktop environments.

Further reading

http://haskell.org/haskellwiki/Grapefruit

4.8.3 Gtk2Hs

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

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

GUIs written using Gtk2Hs use themes to resemble the native look on Windows and, of course, various desktops on Linux, Solaris and FreeBSD. Gtk+ and Gtk2Hs also support Mac OS X (it currently uses the X11 server but a native port is in progress – see below).

Gtk2Hs features:

- automatic memory management (unlike some other C/C++ GUI libraries, Gtk+ provides proper support for garbage-collected languages)
- Unicode support
- high quality vector graphics using Cairo
- extensive reference documentation
- an implementation of the "Haskell School of Expression" graphics API
- support for the Glade visual GUI builder
- bindings to some Gnome extensions: GConf, a source code editor widget and a widget that embeds the Mozilla/Firefox rendering engine
- an easy-to-use installer for Windows
- packages for Fedora, Gentoo (→ 7.4.3), Debian and FreeBSD

The Gtk2Hs library is actively maintained and developed. We had a major new release back in July and another update in November. Of particular note is an introductory tutorial that Hans van Thiel and other contributors have been writing over the past few months.

In the medium term we hope to support the new features in Gtk+ 2.10, to improve the signals API. In the longer term we hope to modularise Gtk2Hs and enable it to be built and distributed with Cabal and Hackage. A promising recent development is that Gtk+'s native (non-X11) backend for Mac OS X has got to the point where Gtk2Hs can be built against it and most of the demo programs work.

The two core maintainers have had less time than they would have liked in recent months to work on Gtk2Hs. This has slowed progress on the medium term goals. On the other hand it is a good opportunity for other contributors to get involved. There are plenty of potential coding projects and Hans has been organising people working on the tutorials. We are interested in feedback from people using Gtk2Hs and especially in interesting applications that we can show off on the website.

Further reading

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

4.8.4 VTY

Report by:	Stefan O'Rear

VTY (Virtualized tTY) is a terminal control library, similar to Stefan Wehr's hscurses. However vty is designed to have a much easier to use API; all communication is accomplished using 5 functions (most using only 2), with a simple data type. Code which describes screen images is pure and declarative. Vty supports all generally useful features of the Linux terminal emulator except for palette setting. It is used successfully by Shellac (→ 4.8.1), Yi (→ 6.15), and the author's unpublished HsLife program.

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

Further reading

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

4.9 (Multi-)Media

4.9.1 Programming of Modular Synthesizers

Report by:	George Giorgidze
Status:	Early Development

In this project we develop a purely functional framework for programming modular synthesizers in Haskell using Yampa. A synthesizer, be it a hardware instrument or a pure software implementation, as here, is said to be modular if it provides sound-generating and sound-shaping components that can be interconnected in arbitrary ways.

Basic sound-generating and sound-shaping modules have been implemented, which are already enough to implement one particular instance of our framework; namely an application capable of playing standard MIDI files with respectable performance, using the SoundFont instrument description standard. The application implements subsets of the aforementioned standards.

Source code and binaries are available under the BSD license.

Future plans

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

Further reading

- Source code and related papers are available from:
  http://cs.nott.ac.uk/~ggg/
- Video of the demo presented at Haskell Workshop 2007:
  http://video.google.com/videoplay?docid=-8742804023527878309

4.9.2 Haskore revision

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

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

1. Basic Cabal support.
2. The `Music` data type has been generalized in the style of Hudak's "polymorphic temporal media."
3. The `Music` data type has been made abstract by providing functions that operate on it.
4. The notion of instruments is now very general. There are simple predefined instances of the `Music` data type, where instruments are identified by Strings or General MIDI instruments, but any other custom type is possible, including types with instrument specific parameters.
5. Support for CSound orchestra files has been improved and extended, thus allowing instrument design in a signal-processing manner using Haskell, including feedback and signal processors with multiple outputs.
6. Support for the software synthesizer SuperCollider both in real-time and non-real-time mode through the Haskell interface by Rohan Drape.
7. The AutoTrack project has been adapted and included.

8. Support for infinite `Music` objects is improved. CSound may be fed with infinite music data through a pipe, and an audio file player like Sox can be fed with an audio stream entirely rendered in Haskell. (See Audio Signal Processing project (→ 6.19).)

9. The test suite is based on QuickCheck and HUnit.

10. Currently we separate a package for managing event lists and a package for managing MIDI files, based on it.

Future plans

- Split into a core package and add-ons, as soon as Cabal supports that.
- Generate note sheets, say via Lilypond.
- Allow modulation of instruments similar to the controllers in the MIDI system.
- Connect to other Haskore related projects.

Further reading

- http://www.haskell.org/haskellwiki/Haskore
- http://darcs.haskell.org/haskore/

4.10 Web and XML programming

4.10.1 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 listings, and is used in the next version of Hoogle (→ 5.5.6).

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

Further reading

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

4.10.2 HaXml

Report by:	Malcolm Wallace
Status:	stable, maintained

HaXml provides many facilities for using XML from Haskell. The public stable release is 1.13.2, with support for building via Cabal for ghc-6.6.x.

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

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

Further reading

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

4.10.3 Haskell XML Toolbox

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

Description

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

The Haskell XML Toolbox bases on the ideas of HaXml (→ 4.10.2) and HXML, but introduces a more general approach for processing XML with Haskell. Since release 5.1 there is a new arrow interface similar to the approach taken by HXML. This interface is more flexible than the old filter approach. It is also safer, type checking of combinators becomes possible with the arrow interface.

Features

- Validating XML parser
- Very liberal HTML parser
- XPath support
- Full Unicode support
- Support for XML namespaces
- Flexible arrow interface with type classes for XML filter
- Package support for ghc

- Native Haskell support of HTTP 1.1 and FILE protocol
- HTTP and access via other protocols via external program curl
- Tested with W3C XML validation suite
- Example programs for filter and arrow interface
- Relax NG schema validator based on the arrows interface
- A HXT Cookbook for using the toolbox and the arrow interface
- Basic XSLT support
- darcs repository with current development version (7.2) under http://darcs.fh-wedel.de/hxt

Current Work

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

A second master student's project, the development of a web server called Janus, has been finished in October of 2006. The title is *A Dynamic Webserver with Servlet Functionality in Haskell Representing all Internal Data by Means of XML*. HXT with the arrows interface has been used for processing all internal data of this web server. The Janus server is highly configurable and can be used not only as HTTP server, but for various other server like tasks. The results of this work will be available via a darcs repository in June 2007. Current activity consists of testing, example applications, demos and documentation.

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

Further reading

The Haskell XML Toolbox Web page (http://www.fh-wedel.de/~si/HXmlToolbox/index.html) includes downloads, online API documentation, a cookbook with nontrivial examples of XML processing using arrows and RDF documents, and master thesises describing the design of the toolbox, the DTD validator, the arrow based Relax NG validator and the XSLT system. A getting started tutorial about HXT is avaliable in the Haskell Wiki (http://www.haskell.org/haskellwiki/HXT).

4.10.4 WASH/CGI – Web Authoring System for Haskell

Report by: Peter Thiemann

WASH/CGI is an embedded DSL (read: a Haskell library) for server-side Web scripting based on the purely functional programming language Haskell. Its implementation is based on the portable common gateway interface (CGI) supported by virtually all Web servers. WASH/CGI offers a unique and fully-typed approach to Web scripting. It offers the following features

- complete interactive server-side script in one program
- a monadic, type-safe interface to generating XHTML output
- type-safe compositional approach to specifying form elements; callback-style programming interface for forms
- type-safe interfaces to state with different scopes: interaction, persistent client-side (cookie-style), persistent server-side
- high-level API for reading, writing, and sending email
- documented preprocessor for translating markup in syntax close to XHTML syntax into WASH/HTML

Completed Items are:

- fully cabalized
- WASH server pages with a modified version of Simon Marlow's `hws` web server; the current prototype supports dynamic compilation and loading of WASH source (via Don Stewart's hs-plugins (→ 4.4.3)) as well as the implementation of a session as a continually running server thread
- Transactional interface to server-side variables and to databases. The interface is inspired by the work on STM (software transactional memory), but modified to be useful in the context of web applications. The interface relies on John Goerzens hdbc package and its PostgreSQL driver.

Current work includes

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

Further reading

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

5 Tools

5.1 Foreign Function Interfacing

5.1.1 C→Haskell

Report by:	Manuel Chakravarty
Status:	active

C→Haskell is an interface generator that simplifies the development of Haskell bindings to C libraries. It reads C header files to automate many tedious aspects of interface generation and to minimise the opportunity for introducing errors when translating C declarations to Haskell.

Duncan Coutts has been busy implementing a new C parser that is very closely aligned to gcc's grammar and has been tested on a large pool of open source code.

More information is at http://www.cse.unsw.edu.au/~chak/haskell/c2hs/.

5.2 Scanning, Parsing, Analysis

5.2.1 Alex version 2

Report by:	Simon Marlow
Status:	stable, maintained

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

The latest release is version 2.1.0.

Changes in version 2.1.0:

- Alex is now in a Darcs repository (→ 6.13), here: http://cvs.haskell.org/darcs/alex.
- Happy has a new build system, based on Cabal. If you have GHC 6.4.2 or later (or Cabal 1.1.4 or later), then you should be able to build and install Alex on any platform. On Windows, Perl is required in addition to GHC for building, but that is all.
- There was a slight change in the error semantics, to enable more informative error messages.

Further reading

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

5.2.2 Happy

Report by:	Simon Marlow
Status:	stable, maintained

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

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

Further reading

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

5.2.3 SdfMetz

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

SdfMetz supports grammar engineering by calculating grammar metrics and other analyses. Currently it supports four different grammar formalisms (SDF, DMS, Antlr and Bison) from which it calculates size, complexity, structural, and ambiguity metrics. Output is a textual report or in Comma Separated Value format. The additional analyses implemented are visualization, showing the non-singleton levels of the grammar, or printing the grammar graph in DOT format. The definition of all except the ambiguity and the NPath metrics were taken from the paper *A metrics suite for grammar based-software* by James F. Power and Brian A. Malloy. The ambiguity metrics were defined by the tool author exploiting specific aspects of SDF grammars and the NPath metric definition was taken from the paper *NPATH: a measure of execution path complexity and its applications.*

Future plans

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

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

Further reading

The web site of SdfMetz (http://wiki.di.uminho.pt/wiki/bin/view/PURe/SdfMetz) includes tables of metric values for a series of SDF grammar as computed by SdfMetz. The tool is distributed as part of the UMinho Haskell Libraries and Tools.

5.3 Transformations

5.3.1 derive

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

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

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

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

Further reading

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

5.3.2 Term Rewriting Tools written in Haskell

Report by:	Salvador Lucas

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

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

Status

The previous work lead to the following tools:

- MU-TERM: a tool for proving termination of rewriting with replacement restrictions (first version launched on February 2002).

  http://zenon.dsic.upv.es/muterm

  Standalone versions of the tool are available for different platforms (Linux, Mac OS X, Windows). A web-based interface (developed in HAppS) is also available:

  http://zenon.dsic.upv.es/webmuterm

- Debussy: a declarative debugger for OBJ-like languages (first version launched on December 2002).

  http://www.dsic.upv.es/users/elp/debussy

- OnDemandOBJ: A Laboratory for Strategy Annotations (first version launched on January 2003).

  http://www.dsic.upv.es/users/elp/ondemandOBJ

  http://www.dsic.upv.es/users/elp/GVerdi

- GVerdi: A Rule-based System for Web site Verification (first version launched on January 2005).

All these tools have been written in Haskell (mainly developed using Hugs and GHC) and use popular Haskell libraries like HAppS, hxml-0.2, Parsec, RegexpLib98, wxHaskell.

Immediate plans

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

References

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

- Proving Termination of Context-Sensitive Rewriting With MU-TERM B. Alarcón, R. Gutiérrez, J. Iborra, and S. Lucas. Electronic Notes in Theoretical Computer Science, 118:105-115, 2007.
- Abstract Diagnosis of Functional Programs M. Alpuente, M. Comini, S. Escobar, M. Falaschi, and S. Lucas Selected papers of the International Workshop on Logic Based Program Development and Transformation, LOPSTR'02, LNCS 2664:1–16, Springer-Verlag, Berlin, 2003.
- OnDemandOBJ: A Laboratory for Strategy Annotations M. Alpuente, S. Escobar, and S. Lucas 4th International Workshop on Rule-based Programming, RULE'03, Electronic Notes in Theoretical Computer Science, volume 86.2, Elsevier, 2003.
- Connecting Remote Tools: Do it by yourSELF! M. Alpuente and S. Lucas. ERCIM News 61:48–49, April 2005.
- MU-TERM: A Tool for Proving Termination of Context-Sensitive Rewriting S. Lucas 15th International Conference on Rewriting Techniques and Applications, RTA'04, LNCS 3091:200–209, Springer-Verlag, Berlin, 2004.
- A Rule-based System for Web site Verification. Demis Ballis and Javier García-Vivó. 1st International Workshop on Automated Specification and Verification of Web Sites, WWV'05, Valencia (SPAIN). Electronic Notes in Theoretical Computer Science, 157(2):11–17, 2006.

5.3.3 HaRe – The Haskell Refactorer

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

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

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

In order to allow users to extend HaRe themselves, HaRe includes an API for users to define their own program transformations, together with Haddock (→ 5.5.5) documentation. Please let us know if you are using the API.

There have been some recent developments for adding program slicing techniques to HaRe. These techniques include a refactoring to split functions returning tuples into separate definitions, and to also put them back together again. There have also been some new refactorings added which work on data types: adding a constructor to a data type and converting a data type into a newtype. The immediate aim for the development of HaRe is to support a number of type-based refactorings.

A snapshot of HaRe is available from our webpage, as are recent presentations from the group (including LDTA 05, TFP05, SCAM06), and an overview of recent work from staff, students and interns. Among this is an evaluation of what is required to port the HaRe system to the GHC API (→ 2.1), and a comparative study of refactoring Haskell and Erlang programs.

The final report for the project appears there too, together with an updated refactoring catalogue and the latest snapshot of the system. Huiqing's PhD thesis on refactoring Haskell programs is now available online from our project webpage.

Further reading

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

5.3.4 VooDooM

Report by:	Tiago Miguel Laureano Alves
Maintainer:	Tiago Alves, Paulo Silva
Status:	stable, maintained

VooDooM supports understanding and re-engineering of VDM-SL specifications.

Understanding is accomplished through the extraction and derivation of different kinds of graphs such as type dependency, function dependency and strongly connected components graphs. These graphs can be subject of both visualization (by exporting into DOT format) and metrication (generating CSV or text report).

Re-engineering is supported through the application of transformation rules to the datatypes to obtain an equivalent relational representation. The relational representation can be exported as VDM-SL

datatypes (inserted back into th e original specification) and/or SQL table definitions (can be fed to a relational DBMS).

The first VooDooM prototype, supporting re-engineering, was developed in a student project by Tiago Alves and Paulo Silva. The prototype was further enhanced and continued as an open source project (http://voodoom.sourceforge.net/) in the context of the IKF-P project (Information Knowledge Fusion, http://ikf.sidereus.pt/) by Tiago Alves and finally in the context of a MSc thesis project.

Future plans

It is planned that the re-engineering functionality of VooDooM will be replaced by the one is being developed for the 2LT project (→ 4.7.5), which will add XML and Haskell generation.

Further reading

VooDooM is available from http://voodoom.sourceforge.net/. The implementation of VooDooM makes ample use of strategic programming, using Strafunski, and is described in *Strategic Term Rewriting and Its Application to a VDM-SL to SQL Conversion* (Alves et al., Formal Methods 2005) and in the MSc thesis *VooDooM: Support for understanding and re-engineering of VDM-SL specifications*.

5.4 Testing and Debugging

5.4.1 Haskell Program Coverage

Report by:	Andy Gill
Status:	released, maintained, in active development

Hpc is a tool-kit to record and display Haskell Program Coverage. Hpc includes tools that instrument Haskell programs to record program coverage, run instrumented programs, and display the coverage information obtained.

Hpc provides coverage information of two kinds: source coverage and boolean-control coverage. Source coverage is the extent to which every part of the program was used, measured at three different levels: declarations (both top-level and local), alternatives (among several equations or case branches) and expressions (at every level). Boolean coverage is the extent to which each of the values True and False is obtained in every syntactic boolean context (ie. guard, condition, qualifier).

Hpc displays both kinds of information in two different ways: textual reports with summary statistics (hpc-report) and sources with colour mark-up (hpc-markup). For boolean coverage, there are four possible outcomes for each guard, condition or qualifier: both True and False values occur; only True; only False; never evaluated. In hpc-markup output, highlighting with a yellow background indicates a part of the program that was never evaluated; a green background indicates an always-True expression and a red background indicates an always-False one.

Hpc provides a Haskell-to-Haskell translator as a means for building instrumented binaries for gathering coverage information, and an Hpc option already checked into GHC 6.7 will make gathering coverage over GHC specific Haskell code possible in GHC 6.8.

The file formats use by Hpc are simple and well documented. The intent is that other tools can be quickly built that process coverage information in creative ways.

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

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

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

GHC has been sucessfully bootstrapping using Hpc, and Hpc has already be deployed internally in Galois in a number of places. In the future expect to see tighter integration between Haskell testing tools and Hpc as obtaining coverage results for test runs becomes standard practice in Haskell development.

Further reading

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

5.4.2 Hat

Report by:	Olaf Chitil and Malcolm Wallace
Status:	maintenance

The Haskell tracing system Hat is based on the idea that a specially compiled Haskell program generates a trace file alongside its computation. This trace can be viewed in various ways with several tools: hat-observe, hat-trail, hat-detect, hat-delta, hat-explore, hat-cover, hat-anim, black-hat, hat-nonterm ... Some views are similar to classical debuggers for imperative languages, some are specific to lazy functional language features or particular types of bugs. All tools inter-operate and use a similar command syntax.

Hat can be used both with nhc98 and ghc (→ 2.1). Hat was built for tracing Haskell 98 programs, but it also supports some language extensions (FFI, MPTC, fundeps, hierarchical libs). A tutorial explains how to generate traces, how to explore them, and how they help to debug Haskell programs.

During the last half year only small bug fixes were committed to the darcs (→ 6.13) repository, but several other updates are also planned for the near future, including new and improved trace-browsers.

Further reading

- A Theory of Tracing Pure Functional Programs
  http://www.cs.kent.ac.uk/~oc/traceTheory.html
- http://www.haskell.org/hat
- darcs get http://darcs.haskell.org/hat

5.4.3 Lazy SmallCheck

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

If there is any case in which a program fails, there is almost always a simple one, and the simplest cases are the easiest to investigate. Such observations motivate the development of SmallCheck, a library that tests program properties for all fully-defined values up to some size. We have developed a variant called Lazy SmallCheck, which generates partially-defined inputs that are progressively refined as demanded by the property under test. The key observation is that if a property evaluates to True or False for a partially-defined input then it would also do so for all refinements of that input. By not generating such refinements, Lazy SmallCheck may test the same input-space as SmallCheck using significantly fewer tests, allowing larger input spaces to be checked in a given amount of time.

A talk about Lazy SmallCheck was given at Fun in the Afternoon at York, and the slides are available, along with our initial implementation, on the Lazy SmallCheck homepage. In the coming months we hope to incorporate further search reduction techniques, include more of SmallCheck's features (e.g. function generation), and explore more examples.

Further reading

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

5.5 Development

5.5.1 Haskell Mode Plugins for Vim

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

There is no standard Haskell mode for Vim, but numerous Vimmers with their own personalized Haskell mode settings for Vim have the kind of IDE functionality at their fingertips that other Haskellers are still looking for.

My own Haskell mode plugins for Vim seem to have become increasingly popular over recent months and collect several scripts that offer functionality based on GHCi, on Haddock-generated documentation, and on Vim's own configurable program editing support. This includes several insert mode completions (based on imported or documented identifiers, on tag files, or on words appearing in current and imported sources), quickfix mode (list errors, jump to error locations), inferred type tooltips, various editing helpers (insert import statement, type declaration or module qualifier for id under cursor, expand implicit into explicit import statement, add option and language pragmas, ..), and direct access to the Haddocks for id under cursor.

Surprisingly many Haskellers are not quite aware of Vim's IDE functions, so I created a little introductory (and incomplete) tour of screenshots giving an incomplete overview of what is available (for more general information, see Vim's excellent built-in `:help`, or browse the help files online at http://vimdoc.sourceforge.net/htmldoc/usr_toc.html; for more and current details of haskell mode features, see the `haskellmode.txt` help file at the project site).

Other Haskell-related plugins for Vim exist – please add links to your own tricks and tips at haskell.org (syntax-colouring works out of the box, other scripts deal with indentation, ..., perhaps there should be a top-level 'Haskell modes for Vim' section at haskell.org, similar to the 'Haskell mode for Emacs' section). I hope these plugins might be useful to some of you (please let me know if anything doesn't work as advertized!), and might even motivate some of you to give Vim a try. It is really not as if Vim (or Emacs, for that matter) didn't have more IDE functionality than most of us ever use, it is more that there is so much of it to learn and to fine-tune to your personal preferences.

Further reading

- Haskell Mode Plugins for Vim: http://www.cs.kent.ac.uk/~cr3/toolbox/haskell/Vim/
- A short tour of some Vim support for Haskell editing (screenshots): http://www.cs.kent.ac.uk/~cr3/toolbox/haskell/Vim/vim.html
- haskell.org section listing these and other Vim files: http://www.haskell.org/haskellwiki/Libraries_and_tools/Program_development#Vim

5.5.2 cpphs

Report by:	Malcolm Wallace
Status:	stable, maintained

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

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

Current release is 1.4, containing some minor bugfixes, especially to macro expansions in cpp conditionals.

Further reading

http://haskell.org/cpphs

5.5.3 Visual Haskell

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

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

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

The sources are in a darcs (→6.13) repository here: http://darcs.haskell.org/vshaskell/, and are provided with a BSD-license. Why not take a look and see what lengths you have to go to in order to write Haskell code that plugs into Visual Studio!

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

5.5.4 EclipseFP – Haskell support for the Eclipse IDE

Report by:	Leif Frenzel
Status:	working, though alpha

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

The aim is to develop an IDE for Haskell that provides the set of features and the user experience known from the Eclipse Java IDE (the flagship of the Eclipse project), and integrates a broad range of Haskell development tools. Long-term goals include support for language-aware IDE features, like refactoring and structural search. The current version is 0.10.

Since the beginning of the year, a new subproject called Cohatoe has developed a framework that allows to partially implement Eclipse Plugins in Haskell. Building on this framework, an EclipseFP 2 branch has been opened where EclipseFP functionality is gradually re-implemented in Haskell, and new functionality is added that integrates existing Haskell tools. Milestone builds from the EclipseFP 2 branch are available for download.

Every help is very welcome, be it in the form of code contributions, docs or tutorials, or just any feedback if you use the IDE. If you want to participate, please subscribe to the development mailing list (see below).

Further reading

- http://eclipsefp.sf.net
- http://lists.sourceforge.net/lists/listinfo/eclipsefp-develop

5.5.5 Haddock

Report by:	Simon Marlow
Status:	stable, maintained

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

The latest release is verison 0.8, released October 10 2006.

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

Changes since the 0.8 release:

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

Further reading

- There is a TODO list of outstanding bugs and missing features, which can be found here: http://darcs.haskell.org/haddock/TODO
- Haddock's home page is here: http://www.haskell.org/haddock/

5.5.6 Hoogle – Haskell API Search

Report by:	Neil Mitchell
Status:	v3.0

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

Hoogle is still under development, but progress is slow due to the author also trying to write a PhD. Hoogle is available as a web interface, a command line tool and a lambdabot (→ 6.14) plugin.

Further reading

http://haskell.org/hoogle

6 Applications

6.1 Exercise Assistants

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

At the Open Universiteit Nederland we are building a collection of tools that support students in solving exercises incrementally by checking intermediate steps. All our tools are completely written in Haskell. The distinguishing feature of our tools is the detailed feedback that they provide, on several levels. For example, we have an online exercise assistant that helps to rewrite logical expressions into disjunctive normal form. Students get instant feedback when solving an exercise, and can ask for a hint at any point in the derivation. Other areas covered by our tools are solving linear equations, reducing matrices to echelon normal form, and basic operations on fractions.

The simplest kind of error to deal with are the syntactical errors, for which we use an error correcting parser combinator library. For each exercise domain, we have formulated a set of rewrite rules, as well as a number of unsound (or buggy) rules to catch common mistakes. With these rules we can check all intermediate steps submitted by the user. We also defined strategies for solving the exercises. A strategy dictates in which order the rules have to be applied to reach the solution, and such a strategy takes the form of a context-free grammar. Strategies are a powerful means to report helpful and informative feedback.

For the near future, we have scheduled sessions with students from our university to validate our approach, and to collect information about the usability of our tools. It is our intention to make the online assistants publicly available. In the future we hope to apply generic programming techniques to support exercises from many more, different domains.

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

Further reading

http://ideas.cs.uu.nl/trac

6.2 Lambda Shell

Report by:	Rob Dockins
Status:	beta, maintained

The Lambda Shell is a feature-rich shell environment and command-line tool for evaluating terms of the pure, untyped lambda calculus. The Lambda Shell builds on the shell creation framework Shellac (→ 4.8.1), and showcases most of Shellac's features.

Features of the Lambda Shell include:

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

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

Further reading

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

6.3 xmonad

Report by:	Don Stewart
Status:	active development

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

The new release 0.5 of xmonad is adding reconfiguration in Haskell, without recompilation, for the first time.

Further reading

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

6.4 GenI

Report by:	Eric Kow
Status:	unchanged, pinged

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

Further reading

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

6.5 Roguestar

Report by:	Christopher Lane Hinson
Status:	early development

Roguestar is a science fiction role playing game belonging to the roguelike family of games (e.g., nethack). Implemented features include a terrain generator, line-of-sight detection, and a frontend based on OpenGL. Roguestar is in the early stages of development. It is not yet "fun."

RSAGL, the Roguestar Animation and Graphics Library, embeds a 3D modelling language within Haskell and plans support for an animation language built on a hierarchy of arrow transformers.

Roguestar is licensed under the GNU GPL. RSAGL is licensed under a permissive license.

Further reading

http://roguestar.downstairspeople.org

6.6 mmisar

Report by:	Slawomir Kolodynski
Status:	under development

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

Further reading

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

6.7 Inference Services for Hybrid Logics

Report by:	Carlos Areces, Daniel Gorin, Guillaume Hoffmann

"Hybrid Logic" is a loose term covering a number of logical systems living somewhere between modal and

classical logic. For more information on this languages, see http://hylo.loria.fr

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

6.7.1 HyLoRes

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

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

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

Further reading

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

6.7.2 HTab

Report by:	Carlos Areces, Daniel Gorin, Guillaume Hoffmann
Status:	active delopment
Current release:	1.2.2

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

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

Further reading

- Hoffmann, G. and Areces, C. *HTab: a terminating tableaux system for hybrid logic*. In Methods for Modalities 5, Cachan, France, 2007.
- Site and source: http://hylo.loria.fr/intohylo/htab.php

6.7.3 HGen

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

HGen is a random CNF (conjunctive normal form) generator of formulas for different hybrid logics. It is highly parametrized to obtain tests of different complexity for the different languages. It has been extensively used in the development of HyLoRes (→ 6.7.1) and HTab (→ 6.7.2).

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

Further reading

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

6.8 Saoithín: a 2nd-order proof assistant

Report by:	Andrew Butterfield
Status:	ongoing

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

Saothín is implemented in GHC 6.4 and wxHaskell 0.9.4, and has been tested on a range of Windows platforms (98/XP/Vista), and should work in principle on Linux/Mac OS X. A first public release of the software in some form is anticipated in early 2008.

Further reading

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

6.9 Raskell

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

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

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

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

Further reading

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

Contact

⟨alex@ittc.ku.edu⟩

6.10 photoname

Report by:	Dino Morelli
Status:	stable, maintained

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

Further reading

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

6.11 HJS – Haskell Javascript Interpreter

Report by:	Mark Wassell
Status:	in development

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

Further reading

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

6.12 FreeArc

Report by:	Bulat Ziganshin
Status:	beta

At this moment, FreeArc is the best practical archiver in the world, providing the maximum speed/compression ratio (http://www.maximumcompression.com/data/summary_mf2.php).

Besides this, FreeArc provides a lot of features, including solid archives with fast updates, tunable compression algorithms, support for external compressors, automatic selection of compression algorithm depending on file type, data encryption and recovery, Win32

and Linux versions, tunable sorting and grouping of files and FAR/Total Commander MultiArc support.

Such an ambitious goal was accomplished by bringing together Haskell and C++: speed-critical parts (compression, encryption) are written in C++ while everything else benefits from fast development and high reliability opportunities provided by Haskell. I should also note that compared to other archivers (traditionally written in C++) FreeArc provides smarter algorithms of archive management which is again due to high level of Haskell programming paradigm.

Program sources are open so you can borrow there:

- compression libraries which includes 11 compression algorithms with easy Haskell interface (http://haskell.org/haskellwiki/Library/Compression)
- encryption code which provides AES, Blowfish, Twofish and Serpent encryption algorithms with all the bells and whistles (PRNG, PBKDF, SHA512) required for really secure encryption of data streams

Moreover, the program includes a few more modules that you may reuse in your program on BSD3 license:

- Win32Files.hs – implements I/O on Windows for files > 4GB and files with Unicode names
- Files.hs – provides an OS-independent interface to the features of Win32Files
- Charsets.hs – encode/decode data in OEM, ANSI, UTF-8/16/32 encodings
- ByteStream.hs – binary serialization library
- UTF8Z.hs – UTF8-packed strings (like ByteString, but with a more memory-efficient representation)
- Process.hs – allows to construct data-processing algorithms from individual processes by joining them together very much like ordinary programs are joined by Unix shell

The program is extensively commented in Russian, so for Russian-speaking Haskellers it may be an invaluable source for learning "practical Haskell".

Further reading

- Download: http://freearc.sourceforge.net

Contact

⟨Bulat.Ziganshin@gmail.com⟩

6.13 Darcs

Report by:	David Roundy
Status:	active development

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

Work has been proceeding on the development of darcs-2, the next major release. We expect that before Christmas this year, the darcs-2 framwork will be integrated into the unstable branch of darcs and ready for experimentation by bold users. At this point, we will be looking for testing by a wide variety of interested users. It will also be an excellent time for new darcs developers to join the fold, as we'll have a fresh start in many ways. In particular, somewhat a flexible patch semantics will lead to the possibility of interesting new features, such as a version of amend-record that will work on older patches that are depended upon by other patches.

Related to this new release, we would all like to thank Eric Kow for the excellent job he has done as the maintainer of the darcs unstable branch. He will be stepping down from this role, but continuing to contribute in the form of patches. David Roundy will be once again stepping in as maintainer of the unstable branch, and will thus be in position to shepherd in the darcs-2 release.

Also expect a ghc-6.8-compatibility release of darcs soon. Patches great and small would be heartily welcome!

Darcs is free software licensed under the GNU GPL.

Further reading

http://darcs.net

6.14 lambdabot

Report by:	Don Stewart
Status:	active development

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

Maintainence for lambdabot continues, along with new plugins and new contributors.

Further reading

- Documentation can be found at:

http://www.cse.unsw.edu.au/~dons/lambdabot.html
- The source repository is available:
  `darcs get`
  http://code.haskell.org/lambdabot

6.15 yi

Report by:	Don Stewart
Status:	active development

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

Yi activity has increased dramatically in the past few months, as Jean-Philippe Bernardy has taken over active development. In particular, Yi is now based on top of the GHC-api (→ 2.1) library, enabling more interactive and dynamic configuration and extension. Significant architectural changes have occurred, including: Vty frontend (→ 4.8.4) replaces Curses frontend; linewrap support; GTK frontend (→ 4.8.3); dynamic Haskell evaluation (like elisp!); new commands may be dynamically defined; Syntax highlighting in GTK frontend.

Further reading

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

6.16 INblobs – Interaction Nets interpreter

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

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

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

The tool is being developed using the repository system Darcs (→ 6.13).

New features

- easier implementation of new reduction strategies
- automatic transformation of lambda terms into interaction nets
- generation of textual descriptions allowing the use of INblobs as a editor/frontend for textual IN compilers
- allow creation of properties' checks
- *Valid IN System* check
- minor changes for better usability

Current Work

A new plugin that will allow INblobs to compile an textual functional program into Interaction Nets is being developed.

Further reading

- Homepage:
  http://haskell.di.uminho.pt/jmvilaca/INblobs/
- Blobs:
  http://www.cs.york.ac.uk/fp/darcs/Blobs

6.17 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.

I am currently preparing a new release (1.13) that will bring compatibility with GHC 6.8, with Cabal 1.2 (→ 4.1.1), and hopefully contain better support for Windows. Help for testing and improving the Windows version is welcome. Current snapshots are available from the Subversion link below.

I would still like to collect some examples of lhs2TeX formatting capabilities and create a gallery on the homepage. If you have written a document that demonstrates nicely what lhs2TeX can do, or if you have designed clever formatting instructions to trick lhs2TeX into doing things previously deemed impossible, please contact me.

Further reading

- http://www.cs.uu.nl/~andres/lhs2tex
- https://svn.cs.uu.nl:12443/viewcvs/lhs2TeX/lhs2TeX/trunk/

6.18 Emping

Report by: Hans van Thiel

Emping is a utility that reads a table of nominal data, in a csv format that can be generated from Open Office Calc, derives all shortest rules for a selected attribute, and writes them to a .csv file that can be read by OO Calc. These rules are partially ordered, and in Emping 0.3 all inference chains can also be shown in OO Calc. A Cabal package (including documentation) is available on Hackage (→ 4.1.1). The next step in the development of Emping is to display the poset of rules (for a selected attribute) in a visual graph. But this is still in the early stages.

Also see http://j-van-thiel.speedlinq.nl/ for the user guide, two white papers, and downloads.

The connection with Haskell is just that it's written in it. But in Haskell it takes only 900 lines of source code in 6 modules (18 pages of printed text, including comments) to implement. Replacing a badly scalable function in version 0.2 with a better one (same signature) in 0.3 proved, as expected, to be absolutely safe.

6.19 Audio signal processing

Report by: Henning Thielemann
Status: experimental, active development

In this project audio signals are processed using pure Haskell code. The highlights are

- a simple signal synthesis backend for Haskore (→ 4.9.2),
- experimental structures for filter networks,
- basic audio signal processing including some hard-coded frequency filters,
- advanced framework for signal processing supported by physical units, that is, the plain data can be stored in a very simple number format, even fixed point numbers, but the sampling parameters rate and amplitude can be complex types, like numbers with physical units,
- frameworks for inference of sample rate and amplitude, that is, sampling rate and amplitude can be omitted in most parts of a signal processing expression. They are inferred automatically, just as types are inferred in Haskell's type system. Although the inference of signal parameters needs some preprocessing, the frameworks preserve the functional style of programming and do not need Arrows and according notation.

  We have checked three approaches, where the last one is the most promising.

  – Explicitly maintain a dictionary of signal parameters in a Reader-Writer-State monad, which must be computed completely before any signal processing takes place. This forces all signal parameters to share the same type and prohibits infinitely many signal processors to be involved (e.g. concatenation of infinitely many short noises).

  – Simulation of logic programming by lazy cycles of function applications (i.e. tied knots, fixed points). The main problems are quadratical computation complexity and a cumbersome and error-prone application. Namely for each input you have to handle a parameter output, and vice versa for propagation of parameters through the network. You need combinators (infix operators) for combining these functions, but you will easily run into cases where you must plug manually, which is a nightmare.

  – Unify only the sample rate. Use a Reader functor / monad. Amplitude is propagated from inputs to outputs only. This is a bit conservative, but fulfills our needs so far.
- We sketched a fusion framework that is specialised to common signal processing routines.

The library comes with basic Cabal support and requires the Numeric Prelude framework (→ 4.6.6) of revised numeric type classes.

Future plans

- Design a common API to the Haskell synthesizer code, CSound support included in Haskore (→ 4.9.2), and the SuperCollider interface.
- Connect with the HaskellDSP library http://haskelldsp.sourceforge.net/. (As a beginning we have prepared and uploaded it to Hackage (→ 4.1.1).)
- Hope on faster code generated by Haskell compilers. :-) Tests with the FastPackedString and the Binary libraries where not promising. It seems that their fusion doesn't handle our cases. We will certainly need a list structure with chunks of unboxed (Storable) elements.

Further reading

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

6.20 hmp3

Report by:	Don Stewart
Status:	stable, maintained

hmp3 is a curses-based mp3 player frontend to mpg321 and mpg123. It is written in Haskell. It is designed to be simple, fast and robust. It's very stable. hmp3 has been updated to version 1.3, and is available from hackage.

Further reading

- Documentation can be found at:
  http://www.cse.unsw.edu.au/~dons/hmp3.html
- The source repository is available:
  `darcs get`
  http://www.cse.unsw.edu.au/~dons/code/hmp3/

6.21 easyVision

Report by:	Alberto Ruiz
Status:	experimental, active development

The *easyVision* project is a collection of libraries for elementary computer vision and image processing applications. We take advantage of Haskell's expressive power without any performance loss, since all heavy numerical computations are done by optimized libraries: HOpenGL for 3D graphics and user interface, GSL-Haskell (→ 4.2.4) for matrix computations, and an experimental binding to Intel's IPP for fast image processing. We use MPlayer for real time image grabbing and video decoding.

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

Recent developments include support for more IPP functions, a simpler way to manipulate state in the applications, some useful camera combinators, and additional illustrative examples.

Further reading

http://alberrto.googlepages.com/easyvision

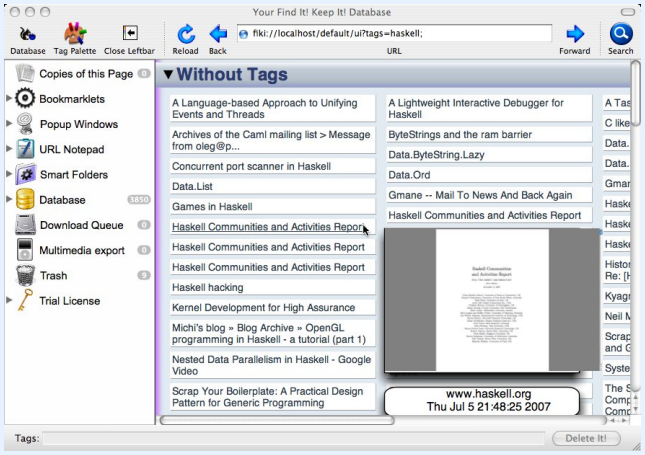
7 Users

7.1 Commercial users

7.1.1 Ansemond LLC

Report by: Sengan Baring-Gould

Find It! Keep It! is a Mac Web Browser that lets you keep the pages you visit in a database. A list of these pages is shown in the "database view". This view is rendered by the browser from generated HTML and is dynamically updated by Javascript DOM operations: tens of thousands of elements cannot efficiently be placed on the screen using DOM operations only, while rerendering a half a megabyte of HTML each time a user interface element changes is unresponsive. A glitch free user experience requires keeping these two separate mechanisms synchronized which proved difficult.



A new Haskell implementation generates abstract DOM operations which are then either rendered to HTML or are converted to Javascript DOM operations to be run within the browser. While this process is not complex (difficult algorithm) it is complicated (hard for the programmer to keep everything in mind). The additional modularity afforded by laziness proved invaluable, enabling all the different pieces to be coded much more independently and clearly than was possible in the original Python version. This same engine could be used on a web server and would work with any web browser. The Haskell version is scheduled to ship in version 1.1 of *Find It! Keep It!*

Ansemond LLC is at http://www.ansemond.com

7.1.2 Barclays Capital Quantitative Analytics Group

Report by: Simon Frankau

Barclays Capital's Quantitative Analytics group is using Haskell to develop an embedded domain-specific functional language (called 'FPF') which is used to specify exotic equity derivatives. These derivatives, which are naturally best described in terms of mathematical functions, and constructed compositionally, map well to being expressed in an embedded functional language. This language is now regularly being used by people who had no previous functional language experience.

The FPF description then serves as the core description of the trade structure, with different interpretations being used to perform operations from generating pricing instructions for the bank's risk systems through to generating trade input forms and reports. The system thus automates the introduction of new products, replacing the previous approach of manually extending each subsystem to cope with a new trade type manually. It has dramatically reduced the turnaround time to make a new trade production-ready.

We have found Haskell to be a great language in which to implement an embedded functional language, and to be very effective as a language for rapid development.

We've been working on the system for a little over a year, and it has been in production use for most of that time, with new features and interpretations being added incrementally over that time.

We are hiring. Please contact Simon Frankau ⟨Simon.Frankau@barcap.com⟩ for more details.

7.1.3 Credit Suisse Global Modelling and Analytics Group

Report by: Ganesh Sittampalam

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

Many of GMAG's models use Excel combined with C addin code to carry out complex numerical computations and to manipulate data structures. This combination allows modellers to combine the flexibility of Excel with the performance of compiled code, but there are significant drawbacks: Excel does not support higher-order functions and has a rather limited and idiosyn-

cratic type system. It is also extremely difficult to use make reusable components out of spreadsheets or subject them to meaningful version control.

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

So far, we have:

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

Current projects include:

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

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

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

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

Further reading

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

7.1.4 Bluespec tools for design of complex chips

Report by:	Rishiyur Nikhil
Status:	Commercial product

Bluespec, Inc. provides tools for chip design, modeling and verification (ASICs and FPGAs), inspired by Haskell and Term Rewriting Systems. Bluespec also uses Haskell to implement many of its tools (over 100K lines of Haskell). Bluespec's products include synthesis, simulation and other tools for Bluespec SystemVerilog (BSV).

BSV is based on the following semantic model: hardware behavior is expressed using *Rewrite Rules*, and inter-module communication is expressed using *Rule-based Interface Methods* (which allow rules to be composed from fragments that span module boundaries). Because rules are atomic transactions, they eliminate a majority of the "timing errors" and "race conditions" that plague current hardware design using existing RTL languages like Verilog or VHDL. Rules also enable powerful reasoning about the functional correctness of systems. In other words, the concurrency model provided by rules is much more powerful and abstract than the low-level concurrency models provided by Verilog, VHDL and SystemC.

BSV incorporates Haskell-style polymorphism and overloading (typeclasses) into SystemVerilog's type system. BSV also treats modules, interfaces, rules, functions, etc. as first-class objects, permitting very powerful static elaboration (including recursion).

Bluespec tools synthesize source code into clocked synchronous hardware descriptions (in Verilog RTL) that can be simulated or further synthesized to netlists using industry-standard tools. This automates the generation of control logic to manage complex concurrent state update, a major source of errors in current design methodology where this logic must be manually coded by the designer.

The powerful Haskell-like static elaboration in BSV is *control adaptive*, i.e., a parameterized design can elaborate into different microarchitectures which have different resource conflicts, but the typically complex control logic necessary to manage these conflicts is synthesized automatically based on the atomic transaction semantics – this is key to BSV's high level of abstraction.

Bluespec participates in standards committees like IEEE P1800 (SystemVerilog) and IEEE P1666 (SystemC), where it tries to encourage adoption of the declarative programming ideas in BSV. One success has been the adoption of Bluespec's proposals for "tagged unions (algebraic types) and pattern matching" in the current IEEE SystemVerilog standard.

Status Bluespec SystemVerilog and its tools have been available since 2004. The tools are now in use by several major semiconductor companies (see Bluespec website or contact Bluespec for details) and several universities (including MIT, CMU, UT Austin, Virginia Tech, Indian Institute of Science, and U.Tokyo).

Availability Bluespec SystemVerilog tools are sold commercially by Bluespec, Inc. Bluespec, Inc. also makes all its tools available at no charge to academic institutions for teaching and research.

Some historical notes and acknowledgements The technology for synthesizing from Term Rewriting Systems to competitive RTL was originally developed by James Hoe and Prof. Arvind at MIT in the late 1990s. At Sandburst Corp., during 2000–2003, Lennart Augustsson was the principal designer of "Bluespec Classic", the first "industrial strength" variant of the lan-

guage, with Rishiyur Nikhil, Joe Stoy, Mieszko Lis and Jacob Schwartz contributing to language and tool development and use. The latter four continued work on BSV at Bluespec, Inc. from 2003 with additional contributions from Ravi Nanavati, Ed Czeck, Don Baltus, Jeff Newbern, Elliot Mednick and several summer interns.

Further reading

- Company website and wiki:
  http://www.bluespec.com http://www.bluespec.com/wiki
- Publications:
  http://www.bluespec.com/technology/research.htm
  Bringing Declarative Programming into a Commercial Tool for Developing Integrated Circuits, Rishiyur Nikhil, Commercial Users of Functional Programming (CUFP), September 2006, slides of presentation at http://www.galois.com/cufp/
  MIT courseware, "Complex Digital Systems":
  http://csg.csail.mit.edu/6.375 and
  http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-884Spring-2005/CourseHome/index.htm
  CMU courseware, "Hardware Systems Engineering":
  http://www.ece.cmu.edu/~ece744
  A fun example with lots of functional-programming features–BluDACu, a parameterized Bluespec hardware implementation of Sudoku:
  http://www.bluespec.com/products/BluDACu.htm

7.1.5 Galois, Inc.

Report by: Andy Adams-Moran

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

Galois develops software under contract, and every project (bar three) that we have ever done has used Haskell. The exceptions used ACL2, Poly-ML, SML-NJ and OCaml, respectively, so functional programming languages and formal methods are clearly our "secret sauce". We deliver applications and tools to clients in industry and the U.S. government. Some diverse examples: Cryptol, a domain-specific language for cryptography (with an interpreter and a compiler, with multiple targets); a GUI debugger for a specialized microprocessor; a specialized, high assurance web server, file store, and wiki for use in secure environments, and numerous smaller research projects that focus on taking cutting-edge ideas from the programming language and formal methods community and applying them to real world problems.

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

Further reading

http://www.galois.com/.

7.1.6 SeeReason Partners, LLC

Report by: Clifford Beshers

Clifford Beshers, David Fox and Jeremy Shaw have formed SeeReason Partners, LLC. Our plan is to deliver services over the internet, using Haskell to build our applications whenever possible. We have chosen primary mathematics skills as our domain, seeking to create a social networking site with games and activities that are both fun and educational.

Often such projects employ large teams of developers and artists to hand-craft a look and feel. Whenever possible, we want to exploit functional programming to generate content or at least distribute the labor more efficiently. Our primary content delivery platform will be Macromedia Flash. We are working on a Haskell to Flash compiler, as well as libraries to construct applications with Adobe's Flex UI toolkit.

Formerly core members of the operating systems group at Linspire, Inc, we continue to maintain the tools for managing a Debian Linux distribution that we developed there. Source code for these tools can be found at our public source code repository http://src.seereason.org/. We plan use these tools to provide current archives of Haskell related packages.

We can be reached at ⟨(cliff,david,jeremy)@seereason.org⟩ and on #haskell (→ 1.2) respectively as thetallguy, dsfox and stepcut.

7.2 Haskell in Education

7.3 Research Groups

7.3.1 Foundations and Methods Group at Trinity College Dublin

Report by: Andrew Butterfield
Participants: Andrew Butterfield, Glenn Strong, Hugh Gibbons, Edsko de Vries

The Foundations and Methods Group focusses on formal methods, category theory and functional programming as the obvious implementation method. A subgroup focusses on the use, semantics and development of functional languages covering such areas as:

- Supporting OO-Design technique for functional programmers
- Using functional programs as invariants in imperative programming
- Developing a GUI-based 2nd-order equational theorem prover (→ 6.8)
- New approaches to uniqueness typing, applicable to Hindley-Milner style type-inferencing.

Recent work in this area included:

- Formal aspects of Functional I/O
- Using Testing to Debug Formal Models

Members of other research groups at TCD have also used Haskell, such as the work done on Image rendering using GHC/OpenGL, in the Interaction, Simulation and Graphics Lab.

Further reading

https://www.cs.tcd.ie/research_groups/fmg/moin.cgi/FunctionalProgramming

7.3.2 Functional Programming at University of Nottingham

Report by: Liyang HU *et al.*

Nottingham is perhaps unique in the UK in bringing together functional programming, type theory and category theory to tackle fundamental issues in program construction. With a total of 20 people in the area, we have a spectrum of interests:

Application of Category Theory The Nottingham group is active in applying ideas from category theory to practical problems in functional programming. Mauro Jaskelioff and Neil Ghani are working on modularity for structured operational semantics. (See also the following sections on initial algebra semantics and containers.)

Containers Nottingham is home to the EPSRC grant on *containers*, a semantic model of functional data structures. Neil Ghani, Thorsten Altenkirch, Peter Hancock, Peter Morris and Rawle Prince are working with containers to both write and reason about programs. Peter Morris has recently finished his PhD which used containers as a basis for generic programming with dependent types.

Datatype-Generic Design Patterns Ondrej Rypacek together with Roland Backhouse and Henrik Nilsson are working on formal reasoning about object-oriented designs with emphasis on algebraic and datatype-generic methods. Our goal is a sound mathematical model allowing us to disclose and formalise correspondences between object-oriented and functional programming.

Dependently-Typed Haskell Supported by a Microsoft Research studentship, Robert Reitmeier is working on integrating dependent types in Haskell under the supervision of Thorsten Altenkirch, with advice from Simon Peyton Jones. Together with Nicolas Oury we are designing an alternative dependently-typed intermediate language, influenced by our experiences with Epigram.

Epigram Epigram (→ 3.3.3) is a dependently-typed functional programming language in its second incarnation, implemented in Haskell. With advice from Conor McBride the Epigram team Thorsten Altenkirch, James Chapman, Peter Morris, Wouter Swierstra and Nicolas Oury are working on both practical and theoretical aspects of the language.

Functional Reactive Programming Yampa, the latest Haskell-based implementation of Functional Reactive Programming (FRP), is currently being maintained by Henrik Nilsson. Under his supervision, Neil Sculthorpe is working on efficient scalable implementation techniques for a Yampa-like language, while George Giorgidze is applying the advantages of FRP to non-causal modelling languages. The latter approach is called Functional Hybrid Modelling.

Functional Specifications of Effects Wouter Swierstra and Throsten Altenkirch have been researching pure specifications of several functions in the IO monad. This research has resulted in the Test.IOSpec library (→ 4.2.3), which may be of interest to anyone who wants to debug, reason about, analyse, or test impure code. Besides implementing these ideas in Haskell, the specifications can be made *total* in the richer type theories underlying Epigram (→ 3.3.3), Coq, and Agda 2 (→ 3.3.2).

Initial Algebra Semantics Neil Ghani has, with Patricia Johann, been working on the initial algebra semantics of advanced data types. They showed that there is no need for the generalised folds in the literature as the standard fold from initial algebra semantics, when coupled with Right Kan extensions, is expressive enough. Interestingly, Left Kan extensions can be employed to give an initial algebra semantics for GADTs. They also used the characterisation of initial algebras as limits to

give short cut fusion rules for both nested data types and GADTs.

Quantum Programming Thorsten Altenkirch and Alex Green are working on a Haskell library to interface or simulate a hypothetical quantum computer – the Quantum IO monad. This is related and inspired by earlier work on the implementation of QML with Jonathan Grattage who finished his PhD on the subject in 2006.

Reasoning About Programs Supported by a grant from EPSRC, Graham Hutton, Nils Anders Danielsson, and Diana Fulger have recently started a new project on reasoning about exceptions and interrupts. This project is also closely related to Liyang HU's ongoing research on reasoning about concurrent systems using software transactional memory. During a sabbatical at Galois (→ 7.1.5) in the summer of 2007, Graham Hutton worked with Andy Gill on the worker/wrapper transformation.

Teaching Haskell plays an important role in the undergraduate programme at Nottingham, as well as our China and Malaysia campuses. Modules on offer include Functional Programming, Advanced Functional Programming, Mathematics for Computer Science, Principles of Programming Languages, Compilers, and Computer-Aided Formal Verification, among others.

Programming in Haskell Graham Hutton has recently completed an introductory Haskell textbook (→ 1.5.1), published by Cambridge University Press, 2007.

Events The group in Nottingham plays a leading role in the Midlands Graduate School in the Foundations of Computing Science, the British Colloquium for Theoretical Computer Science, and the Fun in the Afternoon seminar series on functional programming.

FP Lunch Every Friday, Nottingham's functional programmers gather for lunch with helpings of informal, impromptu-style whiteboard talks. Lecturers, PhD students and visitors are invited to discuss recent developments, problems or projects of relevance. We blog summaries of recent talks.

In the afternoon there is an hour-long seminar. We're always keen on speakers in any related areas: do get in touch with Neil Ghani ⟨nxg@cs.nott.ac.uk⟩ if you would like to visit our group. See you there!

Further reading

- Functional Programming at Nottingham: http://fop.cs.nott.ac.uk/fp/
- Epigram: http://e-pig.org/
- Quantum Programming: http://fop.cs.nott.ac.uk/qml/
- Yampa: http://haskell.org/yampa/
- Fun in the Afternoon: http://fop.cs.nott.ac.uk/fun/
- Midlands Graduate School: http://cs.nott.ac.uk/MGS/
- FP Lunch: http://fop.cs.nott.ac.uk/fplunch/

7.3.3 Artificial Intelligence and Software Technology at JWG-University Frankfurt

Report by:	David Sabel
Members:	David Sabel, Manfred Schmidt-Schauß

Equivalence of Call-by-Name and Call-by-Need

Haskell has a call-by-name semantics, but all efficient implementations of Haskell use call-by-need evaluation avoiding multiple evaluation of the same expression. We showed equivalence of call-by-name and call-by-need for a tiny deterministic letrec-calculus and also the correctness of an unrestricted copy-reduction in both calculi. We also proved that our method scales up to extended letrec-calculi with constructors as well as letrec-calculi with a parallel or operator.

Semantics for Haskell extended with direct-call I/O

We introduced the calculus *FUNDIO* which proposes a non-standard way to combine lazy functional languages with Input/Output using non-deterministic constructs. Program equivalence is based on the operational small-step semantics including the Input/Output behavior of reduction sequences. We proved correctness of a considerable set of program transformations. In particular we have shown correctness of several optimizations of evaluation, including strictness optimizations.

We also analyzed the core language of GHC and the program transformations used in the GHC w.r.t. the FUNDIO semantics. After turning off few transformations which are not FUNDIO-correct and those that have not yet been investigated, we have achieved a FUNDIO-compatible modification of GHC which is called *HasFuse*.

HasFuse correctly compiles Haskell programs which make use of `unsafePerformIO` in the common (safe) sense, since problematic optimizations are turned off or performed more restrictively. HasFuse can also compile programs which make use of `unsafePerformIO` in arbitrary contexts, where the semantics is given by FUNDIO. This allows to combine `unsafePerformIO`

with monadic I/O in Haskell, where the result is reliable in the sense that I/O operations will not astonishingly be duplicated.

Semantics and Transformations for Functional Hardware Descriptions

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

Mutual Similarity

In order to achieve more inference rules for equality in call-by-need lambda-calculi *Matthias Mann* has established a soundness (w.r.t. contextual equivalence) proof for mutual similarity in a non-deterministic call-by-need lambda calculus. Moreover, we have shown that Mann's approach scales up well to more expressive call-by-need non-deterministic lambda calculi, i.e. similarity can be used as a co-induction-based proof tool for establishing contextual preorder in a large class of untyped higher-order call-by-need calculi, in particular calculi with constructors, case, let, and non-deterministic choice. The focus of current research are extensions of these calculi with potential applications in Haskell.

Locally Bottom-Avoiding Choice

For modeling concurrent evaluation of functional programs we investigated an extended call-by-need lambda-calculus with McCarthy's non-deterministic `amb`-operator.

We introduced an observational equivalence based on may- and must-termination w.r.t. a fair small step reduction semantics. We proved correctness of several program transformations, in particular partial evaluation using deterministic reductions. With the developed proof tools it appears promising to prove correctness of further program transformations. Currently we are developing an abstract machine for the calculus extended with `amb` and try to show correctness of this machine. Further research is to apply the may- and must-semantics to other concurrent lambda calculi.

Strictness Analysis using Abstract Reduction

The algorithm for strictness analysis using abstract reduction has been implemented at least twice: Once by Nöcker in C for Concurrent Clean and on the other hand by Schütz in Haskell in 1994. In 2005 we proved correctness of the algorithm by using a call-by-need lambda-calculus as a semantic basis. The proof is published in the Journal of Functional Programming as a forthcoming article appearing 2008.

Most implementations of strictness analysis use set constants like $\top$ (all expressions) or $\bot$ (expressions that have no weak head normal form). We have shown that the subset relationship problem of co-inductively defined set constants is in DEXPTIME.

Further reading

- Chair for Artificial Intelligence and Software Technology
  http://www.ki.informatik.uni-frankfurt.de
- References to all mentioned research topics are collected on the following webpage
  http://www.ki.informatik.uni-frankfurt.de/research/HCAR.html

7.3.4 Formal Methods at Bremen University and DFKI Lab Bremen

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

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

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

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

The Coalgebraic Logic Satisfiability Solver CoLoSS is being implemented jointly at DFKI-Lab Bremen and at the Department of Computing, Imperial College London. The tool is generic over representations of the syntax and semantics of certain modal logics; it uses the Haskell class mechanism, including multi-parameter type classes with functional dependencies, extensively to handle the generic aspects.

Other extensions, libraries and tools of the Glasgow Haskell Compiler, that we exploit include concurrency, existential and dynamic types, Template Haskell, DriFT, Haddock (→ 5.5.5), Programmatica, Shellac (→ 4.8.1), HaXml (→ 4.10.2) and hxt. We also maintain old sources such as bindings to uDrawGraph (formerly Davinci) and Tcl/TK, Shared Annotated Terms (ATerms), and haifa-lite.

Further reading

- Group activities overview: http://www.informatik.uni-bremen.de/agbkb/forschung/formal_methods/
- CASL specification language: http://www.cofi.info
- Heterogeneous tool set: http://www.dfki.de/sks/hets
- Proof General Kit http://proofgeneral.inf.ed.ac.uk/Kit
- The Coalgebraic Logic Satisfiability Solver CoLoSS http://www.informatik.uni-bremen.de/~lschrode/projects/GenMod http://www.doc.ic.ac.uk/~dirk/COLOSS/

7.3.5 Functional Programming at Brooklyn College, City University of New York

Report by:	Murray Gross

A grant has provided us with 6 new quad-processor machines, which we are currently integrating into our existing Linux/Mosix cluster. When the integration is complete, we will be comparing the performance and behavior of the Brooklyn College version of GpH (→ 3.2.1) and the SMP facility of the latest release of GHC (→ 2.1).

In the area of applications, we are working two AI projects, three-dimensional tic-tac-toe (noughts and crosses), and an extended version of the Sudoku puzzle. We have also begun work on a parallel implentation of Skibinski's quantum simulator, which we intend to use to study Grover's fast search algorithm.

Contact

Murray Gross ⟨magross@its.brooklyn.cuny.edu⟩

7.3.6 Functional Programming at Macquarie University

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

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

- Our alpha version of a port of the yhc (→ 2.2) runtime to Palm OS handhelds is working but going into a hiatus (→ 3.1.2).
- Kate Stefanov's PhD thesis on off-the-shelf compression technology for bytecode-based programs was passed. The main results relevant to Haskell folk are good compression ratios for nhc bytecode using a very simple extension of the LZW algorithm.
- Matt Robert's work on the implementation of Jay's pattern calculus continues, with a focus on formal semantics and comparison with related pattern-based calculi.
- We are beginning work on a language processor generation project that will likely use Haskell-based DSLs as the specification notations.

Further reading

Contact us via email to ⟨plrg@ics.mq.edu.au⟩ or find details on many of our projects at http://www.comp.mq.edu.au/plrg/.

7.3.7 Functional Programming at the University of Kent

Report by:	Olaf Chitil

We are a group of staff and students with shared interests in functional programming. While our work is not limited to Haskell, in particular our interest in Erlang has been growing, Haskell provides a major focus and common language for teaching and research. We are continuously looking for new research students.

Our members pursue a variety of Haskell-related projects, many of which are reported in other sections of this report. Chris Brown continues extending HaRe, the Haskell Refactorer (→ 5.3.3). Nik Sultana submitted his MSc thesis supervised by Simon Thompson on formal verification of Haskell refactorings. Thomas Davie, Yong Luo and Olaf Chitil have been working together with the York functional programming group on developing the Haskell tracer Hat (→ 5.4.2) further. They are looking in particular at extensions and improvements of algorithmic debugging. Olaf Chitil and Frank Huch (University of Kiel) developed an assertion library for Haskell. Olaf Chitil also published a more efficient variant of the pretty printing library PPrint. Keith Hanna is continuing work on Vital, a document-centered programming environment for Haskell, and on Pivotal, a GHC-based implementation of a similar environment. Claus Reinke continues improving his popular Haskell Mode plugins for Vim 5.5.1, has posted a mini-tutorial on getting more out of GHCi 7.5.2, has undusted his old extensible record library 4.6.1 for the recent revival of this topic, and has contributed some

minor improvements to GHCi (in 6.8.2) and ghc-pkg (part in 6.8.2, part pending for HEAD). The Kent Systems Research Group is developing an occam compiler in Haskell (Tock).

Further reading

- FP group: http://www.cs.kent.ac.uk/research/groups/tcs/fp/
- Refactoring Functional Programs: http://www.cs.kent.ac.uk/projects/refactor-fp/
- Hat: http://www.haskell.org/hat/
- Lazy Assertions and Pretty Printing: http://www.cs.kent.ac.uk/~oc
- Vital: http://www.cs.kent.ac.uk/projects/vital/
- Pivotal: http://www.cs.kent.ac.uk/projects/pivotal/
- Tock: https://www.cs.kent.ac.uk/research/groups/sys/wiki/Tock

7.3.8 Programming Languages & Systems at UNSW

Report by: Manuel Chakravarty

The PLS research group at the University of New South Wales, Sydney, has produced a couple of Haskell tools and libraries, including the new high-performance packed string library `Data.ByteString` (→ 4.6.2), the hs-plugins (→ 4.4.3) library for dynamically loaded type-safe plugins, the interface generator C→Haskell (→ 5.1.1), and the dynamic editor Yi (→ 6.15). Moreover, we are contributing to widely used Haskell software, such as GHC (→ 2.1), xmonad (→ 6.3), and lambdabot (→ 6.14).

In cooperation with GHC HQ at Microsoft Research, Cambridge, we introduced the idea of type classes with *associated types*, and with GHC HQ and Martin Sulzmann, from the National University of Singapore, we proposed GHC's new intermediate language System F_C. Associated data types (aka *data type families*) and System F_C are fully implemented in GHC's development version and we are currently implementing associated type synonyms; see http://haskell.org/haskellwiki/GHC/Indexed_types for details.

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

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

7.3.9 Haskell in Romania

Report by: Dan Popa

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

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

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

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

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

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

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

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

Papers were presented in International Conferences like ICMI 45.

7.3.10 SCIence project

Report by: Kevin Hammond

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

It makes essential use of functional programming technology in the form of the GRID-GUM functional programming system for the Grid, which is built on the Glasgow Haskell Compiler. The objective is not the technically infeasible goal of rewriting all these (and more) complex systems in Haskell. Rather, we use GRID-GUM to link components built from each of the symbolic systems to form a coherent heterogeneous

whole. In this way, we hope to achieve what is currently a pious dream for conventional Grid technology, and obtain a significant user base both for GRID-GUM and for Haskell. We are, of course, taking full advantage of Haskell's abilities to compose and link software components at a very high level of abstraction.

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

7.4 User groups

7.4.1 Bay Area Functional Programmers

Report by: Keith Fahlgren

The Bay Area Functional Programmers group held their inaugural meeting in September, giving programmers in the San Francisco Bay using or interested in functional programming and functional programming languages such as Haskell, OCaml, SML, Scala, and Erlang a place to meet, discuss, and learn together. We followed up the first informal meeting with an October talk by Alex Jacobson on HAppS (http://happs.org/). November saw David Pollak present the Scala-based web framework lift (http://liftweb.net). We'll finish our series on functional web frameworks with Yariv Sadan presenting the Erlang framework ErlyWeb (http://erlyweb.org/) in December. Videos & slides are available for all the talks at the BayFP blog, as well as information on how to join the mailing list: http://bayfp.org/blog.

7.4.2 OpenBSD Haskell

Report by: Don Stewart

Haskell support on OpenBSD is now taken over by Matthias Kilian, who has updated GHC and related tools for this platform. OpenBSD support for the GHC head branch continues.

For more information see,
http://ports.openbsd.nu/lang/ghc/

7.4.3 Haskell in Gentoo Linux

Report by: Duncan Coutts, Andres Löh

GHC version 6.8.2 is now in Gentoo and (almost) all of the 60+ Haskell libraries and tools work with it too. This includes the latest versions of all the "extralibs" that were released with GHC 6.8.x. There are also GHC binaries available for amd64, sparc and x86.

The fact that all 60+ packages work is no trivial matter since many of them do not yet have official releases that work with ghc-6.8. The packages in Gentoo that do not yet work with 6.8 are hmake, wxhaskell and hs-plugins. In the case of the last two this is a long standing issue since they have no releases that work with ghc-6.6.1 either.

The current set of packages in portage is: `alex`, `alut`, `arrows`, `binary`, `bzlib`, `c2hs`, `cabal`, `cgi`, `cpphs`, `darcs`, `drift`, `fgl`, `filepath`, `frown`, `glut`, `gtk2hs`, `haddock`, `happy`, `harp`, `haskell-src`, `haskell-src-exts`, `haxml`, `hdbc`, `hdbc-odbc`, `hdbc-postgresql`, `hdbc-sqlite`, `hdoc`, `hmake`, `hscolour`, `hslogger`, `hs-plugins`, `hsql`, `hsql-mysql`, `hsql-odbc`, `hsql-postgresql`, `hsql-sqlite`, `hsshellscript`, `html`, `http`, `hunit`, `hxt`, `iconv`, `lhs2tex`, `missingh`, `mtl`, `network`, `openal`, `opengl`, `parallel`, `quickcheck`, `regex-base`, `regex-compat`, `regex-posix`, `rss`, `stm`, `time`, `uuagc`, `uulib`, `wash`, `wxhaskell`, `x11`, `xhtml`, `xmobar`, `xmonad`, `xmonad-contrib`, `zlib`.

See also:
- http://packages.gentoo.org/category/dev-haskell
- http://packages.gentoo.org/package/dev-lang/ghc

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

There are a further 150 packages in the Haskell overlay at http://haskell.org/haskellwiki/Gentoo. There you can access and test the latest versions of the ebuilds, and send patches using darcs (→ 6.13). The overlay is also available via the Gentoo overlay manager "layman". If you choose to use the overlay then problems should be reported on IRC (`#gentoo-haskell` on freenode), where we coordinate development, or via email ⟨haskell@gentoo.org⟩.

We are having some difficulty deciding which are the more popular packages that would be worth adding to portage. If you have suggestions, please contact us.

7.5 Individuals

7.5.1 Oleg's Mini tutorials and assorted small projects

Report by: Oleg Kiselyov

The collection of various Haskell mini-tutorials and assorted small projects (http://pobox.com/~oleg/ftp/Haskell/) – has received three additions:

Class-parameterized classes, the type-level logarithm, and the higher-order for-loop on types

We show invertible, terminating, 3-place addition, multiplication, and exponentiation relations on type-level

unary, Peano numerals. In these relations *any* two operands determine the third. We also show the invertible factorial relation. This gives us all common arithmetic operations on Peano numerals, including base-n discrete logarithm, the n-th root, and the inverse of factorial. The latter operations and division are defined generically, as inverses of exponentiation, factorial and multiplication, resp. The inversion method can work with any representation of (type-level) numerals, binary or decimal.

The inverter itself is generic: it is a type-class *function*, that is, a type-class parameterized by the type-class to 'invert'. The inverter is a simple *higher-order for-loop on types*. In Haskell98+multi-parameter type classes, classes are already first-class, for all practical purposes.

http://okmij.org/ftp/Haskell/types.html#peano-arithm

Restricted Monads

We show how to attain the gist of the restricted datatype proposal (Hughes, 1999) in Haskell, *now*. We need nothing but *plain* multi-parameter type classes: no functional dependencies, no undecidable instances, let alone more controversial extensions, are required. Restricted monads thus should be implementable in Haskell'.

Restricted monads are the extension of monads that restrict the set of values of monadic actions. For example, to implement MonadPlus more efficiently, one often wishes monadic actions yielded the values whose type is in the class Ord.

We propose a fully backward-compatible extension to the monadic interface. All monads are members of the extended monads, and all existing monadic code should compile as it is. In addition, restricted monads become expressible. The article defines the extended interface with the functions

```
ret2  :: MN2 m a   => a -> m a
fail2 :: MN2 m a   => String -> m a
bind2 :: MN3 m a b => m a -> (a -> m b) -> m b
```

which have exactly the same type as the ordinary monadic operations – only with more general constraints. Because new operations have exactly the same type, one may use them in the regular monadic code (given -fno-implicit-prelude flag) and with the do-notation (cf. 'rebindable syntax' feature). Perhaps one day this more general interface becomes the default one.

http://okmij.org/ftp/Haskell/types.html#restricted-datatypes

Impredicativity and explicit type applications

On a simple example we demonstrate that the type system of Haskell with the common rank-2 extension (not counting the extensions in GHC 6.6+) is already impredicative and permits explicit type applications and a limited form of type abstractions (i.e., big-lambda). A `newtype` containing a polymorphic type effectively permits instantiating polytypes with polytypes. Introductions and eliminations of that newtype's data constructor mark the places of type abstraction and type application.

http://okmij.org/ftp/Haskell/types.html#some-impredicativity

7.5.2 dot.ghci

Report by:	Claus Reinke
Status:	new

GHCi is no longer the impoverished offspring of GHC and Hugs it started out as, and it keeps improving. There are, however, still a few commands that Hugs users miss in GHCi, and with the increasing amounts of information available in GHCi, it can sometimes be difficult to find the interesting bits.

Now, you can add some more feature requests to the ticket tracker and hope that someone will get round to implementing them, or you can get the GHC sources and start contributing code yourself. Or, you can make use of some of the less well known features of GHCi, which allow you to define your own GHCi commands (these are still somewhat awkward to use, but present one of the areas in which GHCi has started to overtake Hugs).

In an email to the haskell-cafe in September, I demonstrated this approach in the form of a mini-tutorial, starting with simple things like platform-independent `:pwd`/`:ls`, then laying the groundwork for more complex commands by defining `:redir <var> <cmd>`, a command that redirects the output of `<cmd>`, binding it to variable `<var>`. Based on `:redir`, we can then define `:grep <pat> <cmd>`, to filter the output of `<cmd>` for a pattern `<pat>` (think of finding the help entries related to breakpoints, or the variants of fold appearing in `:browse Prelude`). Taking some examples from the GHCi ticket tracker and from Hugs' commands, there is also `:find <id>` (open the source for the definition of `<id>`), `:b` (`:browse` first module listed in `:show modules`), and `:le <mod>` (load module `<mod>`, edit location of first error, if any).

The commands are self-documenting, can be listed and removed as a group, and should give a good starting point for your own experiments with GHCi's `:def`. Since the email, which targeted GHC 6.6 and later, a version for GHC 6.4.1 was added for those who needed to work with an old installation (the commands in that version differ slightly, to account for 6.4.1's limitations). Please let me know if you find this useful, and remember to share your own GHCi tricks and tips!

Further reading

- http://www.haskell.org/pipermail/haskell-cafe/2007-September/032260.html
- http://www.cs.kent.ac.uk/~cr3/toolbox/haskell/#dot.ghci

7.6 A Survey on the Use of Haskell in Natural-Language Processing

Report by: Richard A. Frost

The survey "Realization of Natural-Language Interfaces Using Lazy Functional Programming" was published in ACM Computing Surveys Volume 38 Issue 4 Article 11 in December 2006. It was in the Top 10 downloads from ACM Magazines and Surveys for February and March of 2007.

The survey currently contains 168 references to relevant publications. The survey will be updated as more information becomes available. Please send information on recent publications in this area to ⟨rfrost at cogeco.ca⟩

Further reading

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

7.6.1 Inductive Programming

Report by: Lloyd Allison

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

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

There has been some progress since May 2007: Estimators and Models have been expanded to include the Poisson and Geometric distributions. Naturally these can be (and have been) used in and with the existing types, classes, estimators and functions: Models (probability distributions), function-models (regressions), time-series (e.g. Markov models), series segmentation, mixture models, classification trees (and regression trees and model trees), and Bayesian networks. An overhaul of the code remains overdue.

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

Future plans

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

Further reading

http://www.csse.monash.edu.au/~lloyd/tildeFP/II/

7.6.2 Bioinformatics tools

Report by: Ketil Malde

The Haskell bioinformatics library supports working with nucleotide and protein seqeunces, importing Fasta-formatted files with associated quality, and also the TwoBit and PHD sequence formats. BLAST output in XML format is supported, and the standard alignment algorithms (and some non-standard ones) are provided.

It is considered in development (meaning things will be added, some functionality may not be as complete or well documented as one would wish, and so on), but central parts should be fairly well documented and come with a QuickCheck test suite.

The library abstracts functionality that is used in a handful of applications, including:

- `xsact` – an EST clustering program
- `RBR` – a repeat detector/masker
- `clusc` – a tool for calculating cluster similarity with a bunch of metrics
- `dephd` – a sequence quality assessment tool
- `xml2x` – a BLAST postprocessor and GO annotator

Everything is GPLed and available as darcs repos (→ 6.13), at http://malde.org/~ketil/.