Haskell Communities and Activities Report

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

– fifth edition –

November 11, 2003

Claus Reinke (editor), University of Kent, UK
Krasimir Angelov, Bulgaria
Sengan Baring-Gould, AMD, USA
Mark T.B. Carroll, Aetion Technologies LLC, USA
Manuel Chakravarty, University of New South Wales, Australia
Olaf Chitil, The University of York, UK
Iavor Diatchki, OGI School of Science and Engineering, OHSU, USA
Shae Erisson, Sweden
Levent Erkok, OGI School of Science and Engineering, OHSU, USA
Andrew Frank, Institute for Geoinformation, TU Vienna, Austria
Murray Gross, City University of New York, USA
Walter Guttmann, University of Ulm, Germany
Jurriaan Hage, Utrecht University, The Netherlands
Thomas Hallgren, Pacific Software Research Center, OGI/OHSU, USA
Keith Hanna, University of Kent, UK
Johan Jeuring, Utrecht University, The Netherlands
Isaac Jones, Aetion Technologies LLC, USA
Daan Leijen, Utrecht University, The Netherlands
Rita Loogen, University of Marburg, Germany
Christoph Lüth, George Russell, and Christian Maeder, University of Bremen, Germany
Matthias Mann and David Sabel, JWG-University Frankfurt, Germany
Simon Marlow, Microsoft Research Cambridge, UK
Henry Nyström, Heriot-Watt University, Scotland
Rex Page, Oklahoma University, USA
Sven Panne, Germany
Ross Paterson, City University London, UK
Jens Petersen, Red Hat, Japan
John Peterson, Yale University, USA
Simon Peyton Jones, Microsoft Research Cambridge, UK
Bernie Pope, University of Melbourne, Australia
Frank Rosemeier, FernUniversität in Hagen, Germany
Alastair Reid, Reid Consulting (UK) Ltd., UK
David Roundy, MIT, USA
Chris Ryder, University of Kent, UK
Uwe Schmidt, Fachhochschule Wedel, Germany
Sean Seefried, University of New South Wales, Australia
Axel Simon, University of Kent, UK
Ganesh Sittampalam, Oxford University, UK
Anthony Sloane, Macquarie University, Australia
Dominic Steinitz, UK
Doaitse Swierstra, Utrecht University, The Netherlands
Martin Sulzmann and Jeremy Wazny, National University of Singapore, Singapore
Wolfgang Thaller, Graz, Austria
Peter Thiemann, University of Freiburg, Germany
Phil Trinder, Heriot-Watt University, Scotland
Eelco Visser, Utrecht University, The Netherlands
Malcolm Wallace, The University of York, UK
Ashley Yakeley, Seattle WA, USA

Preface

Months of work, hundreds of emails, and weeks of collecting material, all condensed for you to read through over a weekend, and all about Haskell!-) That, in essence, are the Haskell Communities and Activities Reports, now in their fifth edition. Twice a year, they attempt to provide overviews of all things Haskell over the last 6 months, particularly focussing on recent and current activities and plans.

The reports are created by inviting all Haskellers to contribute brief summaries of their recent Haskell activities and collecting all contributions in a single document. In spite of the format, they are basically an email survey of the Haskell mailing list, and **you are invited** to send in your contributions. Please make a note in your diary now: **your entries for the May 2004 edition should reach the editor in the last two weeks of April 2004!** If you come across any interesting Haskell implementations, extensions, libraries, tools, papers, user groups, applications etc. in the coming 6 months, please make sure they are reflected in the next edition, by prompting the people responsible.

What you'll find in this report are updates on recent activities in your favourite Haskell groups and projects, confirmations by authors and maintainers that their software is still actively maintained and, of course, some stuff you may not have come across before!-) In many cases, you will be invited to review the process that has been made in specialist groups, and to provide feedback, or to contribute your time and efforts to keep good projects going or get new projects off the ground. Remember, nearly everything you see here is the result of volunteer efforts, and depends on your contributions.

One example are these reports: more than two years ago, one frustrated Haskell user complained at a Haskell workshop that it had become impossible to follow the developments in the various specialist Haskell lists and groups, and was promptly asked whether he'd be willing to do something about it. Well, he was, and the resulting reports are by now firmly established as a means to keep up to date with increasingly diverse Haskell groups and activities. However, we all have other projects that need our time and, after two years and four editions, this year's Haskell workshop saw me **looking for a successor in the editor role**.

Demonstrating once again that Haskellers are not shy to contribute their efforts, we already have a candidate, *Andres Löh* (thanks!), who is currently studying for his PhD at Utrecht University. So it might seem that **you** will have to volunteer for one of the many other jobs mentioned in this report;-)

However, there's a small hitch, in that Andres is sensible enough not to want to start this in the final phase of his PhD: he will first be available to edit the November 2004 edition. If you read this, the November 2003 edition will finally have made its way out, defeating numerous delays and obstacles, but **we still need someone to take on the May 2004 edition**. If you can bring in a natural curiosity, good email- and text-processing tools, and are willing to invest some of your time for the benefit of the Haskell community, please get in touch with me! It may even be a good idea to have two editors alternating in their duties for the May and November editions, to reduce the impact on any one person's time budget.

Oh, and last but not least, these reports are not just meant for your next weekend: they are also an opportunity to present a glimpse of the strange world you live and work in to outsiders, be it that you want to show your students that there is more to Haskell than obscure type class error messages, or be it that you want to convince your boss that, yes, there are implementations, tools, and libraries, lots of people working on them, and there are companies using Haskell, and there are even companies and consultants specialising in Haskell (we certainly haven't covered them all here).

The most recent edition is also always a good entry point for Haskell newcomers, giving them a chance to find their way around – so if you find yourself answering the same questions on projects, implementations, libraris, tools, status, and plans over and over again, you might find it convenient to direct them towards `http://www.haskell.org/communities/`.

But now sit back and enjoy the read, then follow the pointers and try things out, give the authors feedback on their work, discuss, contribute, collaborate, or start your own little projects and excursions into Haskell land. And, please, remember to come back in 6 months and report!-)

As always, this edition is the result of your work and contributions, and so I'd like to close with a big thanks to all contributors!

Claus Reinke, University of Kent, UK

Contents

Chapter 1

General

1.1 haskell.org

Report by: *John Peterson*

haskell.org belongs to the entire haskell community - we all have a stake in keeping it as useful and up-to-date as possible. Anyone willing to help out at `haskell.org` should contact John Peterson (<peterson-john@cs.yale.edu>) to get access to this machine. There is plenty of space and processing power for just about anything that people would want to do there. What can `haskell.org` do for you?

- advertise your work: whether you're developing a new application, a library, or have written some really good slides for your class you should make sure `haskell.org` has a pointer to your work.
- hosting: if you don't have a stable site to store your work, just ask and you'll own `haskell.org/yourproject`.
- mailing lists: we can set up a mailman-based list for you if you need to email your user community.
- sell merchandise: give us some new art for the cafepress store. publicize your system with a t-shirt.

The biggest problem with `haskell.org` is that it is difficult to keep the information on the site current. At the moment, we make small changes when asked but don't have time for any big projects. Perhaps the biggest problem is that most parts (except the wiki) cannot be updated interactively by the community. There's no easy way to add a new library or project or group or class to haskell.org without bothering the maintainers. the most successful sites are those in which the community can easily keep the content fresh. We would like to do something similar for `haskell.org`.

Just what can you do for `haskell.org`? Here are a few ideas:

- haskell programmers are not graphic designers. just about anyone could make `haskell.org` look nicer and more professional.
- make the site more interactive. allow people to add new libraries, links, papers, or whatever without bothering the maintainers. allow people to attach comments to projects or libraries so others can benefit from your experience. help tell everyone which one of the graphics packages or gui's or whatever is really useful.
- develop a system where the pages for `haskell.org` live in a cvs repository so that we can more easily share out maintenance.
- add searching capability to `haskell.org`.
- take over the cafepress store and get more merchandise in there.

Some of these ideas would be good student projects. Be lazy - get students to do your work for you.

Further reading:

`http://www.haskell.org`
`http://www.haskell.org/mailinglist.html`

1.2 Tips, Tricks, Tours and Tutorials

Alastair Reid continues to add to his guide to the new Foreign Function Interface
`http://www.reid-consulting-uk.ltd.uk/docs/ffi.html`

1.3 Haskell-related Publications

In this section, we try to give pointers to and perhaps short descriptions of recent Haskell-related publications (books, conference proceedings, special issues in journals, PhD theses, etc.), with brief abstracts. For a more exhaustive overview of Haskell publications, see Jim Bender's "Online Bibliography of Haskell Research" (`http://haskell.readscheme.org`). Please make sure to keep him up to date about new (and not so new) Haskell-related publications!
And if you still haven't come across the Haskell bookshelf, you'll find it at `http://www.haskell.org/bookshelf/`. It lists textbooks, papers (especially of tutorial nature), proceedings of the "Advanced Functional Programming" summer and spring schools, as well as reference material, often created in the context of Haskell courses.

In *"Discriminative Sum Types Locate the Source of Type Errors"* (ICFP'03, `http://doi.acm.org/10.1145/944705.944708`), *Neubauer and Thiemann* describe a new approach to locating type errors in Haskell using a non-standard type system which is based on the theory of discriminative sum types. A type derivation in this system contains sufficient information to highlight the program locations that participate in a type error; a browser based on a type inference procedure for that system is currently under development.

Daan Leijen has finished his PhD Thesis at Utrecht University: *"The λ Abroad - A Functional Approach to Software Components"*, November 2003. `http://www.cs.uu.nl/~daan/download/papers/phd-thesis.pdf`

Simon Peyton Jones and Mark Shields are working on *"Practical type inference for arbitrary-rank types"*: This paper, long in gestatation, describes the approach that GHC takes to type inference for higher-rank types. It has a strongly tutorial flavour, and comes with an executable implementation. Feedback on this paper would be very welcome.
`http://research.microsoft.com/Users/simonpj/papers/putting/`

1.3.1 Haskell-related Events

In the following pages, you will see various references to acronyms like ICFP, IFL, HIM, etc., so it might be useful to know about some standard and not so standard events that have featured Haskell-related presentations or publications recently, especially if you want to follow the common practice of going through their programmes and trying to find the papers or slides for the talks that interest you on the author's homepages:

ICFP 2003 – 8th ACM SIGPLAN International Conference on Functional Programming (ICFP 2003) Uppsala, Sweden: 25-29 August 2003.
`http://www-users.cs.york.ac.uk/~colin/icfp2003.html`
Proceedings online in the ACM digital library:
`http://portal.acm.org/toc.cfm?id=944705&type=proceeding&coll=GUIDE&dl=ACM&CFID=13941416&CFTOKEN=10629846`

Haskell Workshop 2003 – ACM SIGPLAN 2003 Haskell Workshop Uppsala, Sweden: August 28, 2003
`http://www.cs.uu.nl/~johanj/HaskellWorkshop/cfp03.html`
Proceedings online in the ACM digital library:
`http://portal.acm.org/toc.cfm?id=871895&type=proceeding&coll=portal&dl=ACM&CFID=13941416&CFTOKEN=10629846`

HIM 2003 – Haskell Implementor's Meeting 2003, Stockholm, Sweden: 29/30 August (by-invitation only, but webpage has slides of several talks you might be interested in):
`http://cs-www.cs.yale.edu/homes/nilsson/HIM/`

IFL 2003 – 15th International Workshop on the Implementation of Functional Languages Edinburgh, Scotland September 8th - 10th, 2003.
`http://www.macs.hw.ac.uk/~ifl03/`

TFP 2003 – Fourth symposium on Trends in Functional Programming September 11th-12th 2003, Edinburgh, Scotland
`http://homepages.inf.ed.ac.uk/stg/workshops/TFP/`

SBLP 2003 – 7th Brazilian Symposium on Programming Languages Ouro Preto, MG, Brazil - May 28-30, 2003
`http://www.inf.pucminas.br/sblp2003/`

LL3 – Lightweight Languages 2003 Nov. 8, 2003 MIT, Cambridge MA featured a number of Haskell offspring this year, such as WASH/CGI (section 4.8.3), the BlueSpec hardware description language, and Functional Reactive Programming, ported to Scheme. Instead of proceedings, there are webcast recordings, and a mailing list with keyword-style notes from the event:
`http://ll3.ai.mit.edu/`

Chapter 2

Implementations

2.1 The Glasgow Haskell Compiler

Report by: *Simon Peyton-Jones*

GHC status (October 2003) Quite a bit is happening on the GHC front. As ever, we are grateful to the many people who submit polite and well-characterised bug reports. We're even more grateful to folk who actually help develop and maintain GHC. The more widely-used GHC becomes, the more Simon M and I rely on you to help solve people's problems, and to maintain and develop the code. We won't be around for ever, so the more people who are involved the better. If you'd like to join in, please let us know.
Here are some highlights from the last few months.

- Simon PJ has been busy re-working chunks of the type-checker, partly as a clean-up sweep, but also to consolidate it for further Template Haskell developments. The big change is that supporting definitions are now sucked from interface files lazily (rather than eagerly as before); this should greatly reduce the quantity of interface files read. The version tracking for separately-compiled modules is also somewhat more refined, reducing the size of interface files.
- Simon M has been working on a Visual Studio plug-in, so that GHC can be hosted in Visual Studio (section 5.3.3). Nothing fancy yet (syntax colouring etc). The next stage is to add much more detailed location information to the syntax tree, something we have been meaning to do for ages.
- With heroic help from Ross Paterson, GHC now supports the syntactic sugar that Ross invented for John Hughes's "arrows". It's more than just sugar: the arrow syntax is type-checked directly, so that errors are reported in terms of arrow syntax, not some complex desugaring thereof (section 3.5.2).
- Sigbjorn Finne has added experimental support for .NET interop.
- Simon M has replaced GHC's hand-written lexer by an Alex-generated lexer. He had to beef up Alex quite a bit to do this (section 5.2.1).
- Quite a bit of work has been done to make GHC easier to port. Donald Stewart and others are hard at work doing 64-bit ports of GHC.
- Ralf Lämmel has extended the "scrap your boilerplate" libraries (Data.Generics) so that they elegantly support 'read' and 'show', which earlier versions did not (cf. sections 3.4 and 4.3.3). Paper forthcoming.

Upcoming excitements:

- We're planning a big clean up in the back end, combining Abstract C with Stix. This is desirable in its own right, but it's also on the path to generating `C--` from GHC, which we want to do now that `C--` is available (`http://www.cminusminus.org`).
- Tim Sheard and Simon are hatching a collection of improvements to Template Haskell (`http://research.microsoft.com/~simonpj/tmp/notes2.ps`). Template Haskell hasn't had much attention over the summer, what with holidays and ICFP.
- We will adapt GHC to work with whatever scheme Isaac comes up with for library packaging (section 4.1.1).

Releases 6.2 is coming out soon, so all the major overhaul referred to will be in 6.4, not 6.2 – no date planned yet.

Further reading:

`http://www.haskell.org/ghc/`

2.2 Hugs

Report by: *Ross Paterson*
Status: *Stable, actively maintained, volunteers welcome*
Hugs is a very portable, easily installed Haskell-98 compliant interpreter that supports a wide range of type-system and runtime-system extensions including typed record extensions, implicit parameters, the foreign function interface extension and the hierarchical module namespace extension.

Current state At the time of writing, a new major release of Hugs is almost ready.
With this release, Hugs will rely exclusively on the Haskell hierarchical libraries. This reduces the amount of Haskell code to be maintained with Hugs, and also increases compatibility with the other implementations. Coverage has also

improved – Hugs now supports imprecise exceptions (but not asynchronous ones), unboxed arrays and more. Compatibility stubs for old libraries are also provided as a transitional measure, but some day these will disappear.

With these library improvements, together with Hugs's longstanding support for various Haskell extensions and the recent addition of FFI support, code developed with GHC can often be made to work with Hugs too with a little effort. Sven Panne has done this with his GLUT and OpenGL packages (section 4.6.4), and we would encourage other developers to do the same.

Interoperation with .NET (on Windows platforms), formerly a separate add-on, has been enhanced and is now integrated with Hugs. You can instantiate and use .NET objects from within Haskell, and call and use Haskell functions from any .NET language.

Assorted fragments of documentation have been re-organized and augmented as a Users's Guide describing the current state of Hugs. It is however less complete than we would like in places. Contributions are welcome.

Future plans Hugs will continue to improve its coverage of the libraries. Older interfaces will disappear.

Sven Panne intends to modernize the configuration system, which is currently creaking with age, and duplicates parts of the fptools configuration system.

The manpower available for Hugs development and maintenance is very limited. Former maintainers Sigbjorn Finne and Jeffrey Lewis are now very busy with other things, but help out when they can. Alastair Reid has also been very busy in the last 6 months.

Contributions from volunteers are welcome. For example, Dimitry Golubovsky <dimitry@golubovsky.org> is working on adding optional Unicode support to Hugs. People who test the CVS version are also a great help.

Further reading:

`http://www.haskell.org/hugs/`

2.3 nhc98

Report by: *Malcolm Wallace*
Project status: *stable, maintained*

nhc98 remains a small, easy to install, standards-compliant compiler for Haskell 98. Its implementation is stable and the public release remains at version 1.16 for the moment. Maintenance and bugfixes continue to the CVS tree at haskell.org. When sufficient serious fixes have accumulated, a new public release will be forthcoming. No innovative new features are currently planned.

Further reading:

`http://www.haskell.org/nhc98/`

2.4 hmake

Report by: *Malcolm Wallace*
Project status: *stable, maintained*

Hmake is an intelligent compilation management tool for Haskell programs. It is stable at public release version 3.07, with occasional maintenance and bugfixes to the CVS tree at haskell.org.

Future plans In the last issue of the HC&A report, it was suggested that hmake should allow the external configuration of source code preprocessors. (Known preprocessors are currently hard-coded.) This is still on the to-do list, together with some other feature enhancements suggested by users. New suggestions always welcome.

Further reading:

`http://www.haskell.org/hmake/`

2.5 Domain-specific variations

2.5.1 Haskell on Handheld Devices

Report by: *Anthony Sloane*

We are making further progress on our port of nhc98 to Palm OS but other activities (section 6.5.4) have slowed us down somewhat. We are in the process of bringing it up to the latest version of nhc98. A paper on this project was presented at the Implementation of Functional Languages workshop this year. Our revised schedule for the public release of a beta version is sometime over the (southern hemisphere) summer.

2.5.2 Helium

Report by: *Daan Leijen*
(Arjan van IJzendoorn, Bastiaan Heeren, Daan Leijen, Rijk-Jan van Haaften)

The purpose of the Helium project is to construct a lightweight compiler for a subset of Haskell that is especially directed to beginning programmers (see *"Helium, for learning Haskell"*, Bastiaan Heeren, Daan Leijen, Arjan van IJzendoorn, Haskell Workshop 2003). We try to give useful feedback for often occurring mistakes. To reach this goal, Helium uses a sophisticated type checker described in section 3.3.2 (see also *"Scripting the type inference process"*, Bastiaan Heeren, Jurriaan Hage and S. Doaitse Swierstra, ICFP 2003). Helium now has a simple graphical user interface that provides online help. We plan to extend this interface to a full fledged learning environment for Haskell. The complete type checker and code generator has been constructed with the attribute grammar (AG) system developed at Utrecht University (section 6.5.5) One of the aspects of the compiler is that it also logs errors, so we can track the kind of problems students are having, and improve the error messages and hints. The compiler uses the LVM (Lazy Virtual Machine) as backend. The LVM uses a portable instruction set and file format that is specifically designed to execute lazy higher-order languages, and is formally described in the PhD thesis of Daan Leijen (section 1.3).

Further reading:

`http://www.cs.uu.nl/research/projects/helium/`

2.5.3 Educational Domain Specific Languages

Report by: *John Peterson*
Project status: *maintained, stable*

The goal of this project is to bring functional programming to users that are not trained computer scientists or programmers. We feel that the simplicity of functional programmiung makes it an ideal way to introduce programming language concepts and encourage a basic literacy in computational principles. Languages can also be used as part of a domain-centered learning experience, allowing functional programming to assist in the instruction of subjects such as mathematics or music.

Our languages are media oriented. They allow students to explore the basic principles of functional programming while creating artifacts such as images, animations, and music.

These languages have been used for high school mathematics education, an introduction to functional programming for students in high school programming classes, and as a gentle way to present functional programming in a programming language survey class. The graphics language, Pan#, runs all of the examples in Conal Elliott's Fun of Programming chapter with only a few minor changes. It also runs many of the examples found in Jerzy Karczmarczuk's Clastic system.

There are two languages under development. The first is Pan#, a port of Conal Elliott's Pan compiler to the C# language. This runs on Windows using .NET and is easy to install and use. This probably would run on Linux using Mono (.NET for other platforms) but we have not attempted this yet. The front end of this system is a mini-Haskell interpreter which is currently somewhat unsophisticated - we plan to customize Helium (section 2.5.2) for this purpose in a future release. Our website contains a number of examples produced by this language and some instructional materials. A new release of Pan# is expected in November, 2003. This will add many new examples, including all of the Fun of Programming programs and many Clastic examples, as well as offer significant improvements to the viewer. Random textures are also included in this new release.

Our second language describes music using Paul Hudak's Haskore system. We are currently re-packaging Haskore to simplify the language somewhat and add a few new capabilities, including support for randomized music. We are currently working on a tutorial for the system and should have a release ready in December 2003.

Further reading:

`http://haskell.org/edsl/`

2.5.4 Vital: Visual Interactive Programming

Report by: *Keith Hanna*
Project status: *new release imminent*

Vital is a Haskell-based, visual environment intended for the interactive, exploratory development of programs by non computer-specialist end-users (engineers, analysts, etc.): each Haskell module is associated with a worksheet on which its declarations and expressions may be located and their values graphically displayed (in a form determined by a type-indexed stylesheet).

The Vital environment embodies the principles of direct manipulation. In particular, it allows the graphical displays to be edited by mouse gesture (for example, the values in an array or the shape of a tree might be changed) with such changes being reflected in the Haskell source code.

A release of a fairly comprehensive implementation of Vital is planned for later this month.

Further reading:

`http://www.cs.kent.ac.uk/projects/vital/`

Chapter 3

Language Extensions

3.1 Foreign Function Interface

Report by: *Manuel Chakravarty*
Project status: *Version 1.0 (RC13)*
Release Candidate 13 of the Haskell 98 FFI Addendum has been released and is, modulo errata, what will become Version 1.0 of the Addendum. The definition is available from
`http://www.cse.unsw.edu.au/~chak/haskell/ffi/`

3.2 Non-sequential Programming

3.2.1 Concurrent Haskell

The discussion about Concurrent Haskell and OS threads has been continued. A solution to the problems posed by integrating lightweight threads, OS threads and foreign libraries that use OS-thread-local state has been presented at the Haskell Implementor's Meeting in Stockholm (slides available at `http://cs-www.cs.yale.edu/homes/nilsson/HIM/Slides/Thaller-BoundThredadsPresentation.pdf`); that mechanism is implemented in the CVS version of GHC 6.4; whether it will be merged into the upcoming GHC 6.2 is yet to be determined. The mechanism is documented in the library documentation for `Control.Concurrent` (in CVS), and there will be a more formal document that will be proposed as an addendum to the Haskell report and FFI addendum.
The Concurrent API is described here:
`http://www.haskell.org/ghc/docs/latest/html/base/Control.Concurrent.html`

3.2.2 GpH – Glasgow Parallel Haskell

Report by: *Phil Trinder*

The Team *Phil Trinder, Kevin Hammond, Hans-Wolfgang Loidl, Abyd Al Zain, Jost Berthold, Xiao Yan Deng, Murray Gross, Andre Rauber du Bois, Alvaro Rebon Portillo, Leonid Timochouk.*

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 modules, to improve performance and support for architecture-independence, are under development.

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

- The stable branch (GUM-4.06, based on GHC-4.06) is available for RedHat-based Linux machines: binary snapshot (`ftp://ftp.macs.hw.ac.uk/pub/gph/`, see README.GUM for installation instructions). The stable branch is also available from the GHC CVS repository via tag `gum-4-06`.
- The unstable branch (GUM-5.02, based on GHC-5.02) is currently being tested on a Beowulf cluster. Most of our test programs run already, with minor issues left to be resolved before this version will become our main development version. The unstable branch is available from the GHC CVS repository via tag `gum-5-02-3`.

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 Sun-Solaris shared-memory machine exists but currently suffers from performance problems, which we are trying to track down.
- A port to a Mosix cluster has been built in the Metis project at Brooklyn College (section 6.5.3), with a first version available on request from Murray Gross.
- The Eden version from the GHC CVS(tag `eden-5-02-3`) supports both GpH and Eden languages (section 3.2.4).

System Evaluation and Enhancement

- We have ported GUM to computational GRIDs, replacing the current PVM layer with IMPICH-G2. We have measured programs on a single grid-enabled cluster, and are very close to measuring programs on more than 1 cluster.
- We are teaching GpH to students at Heriot-Watt `http://www.macs.hw.ac.uk/~trinder/ParDistr/` and Phillips Universitat Marburg `http://www.mathematik.uni-marburg.de/~loogen/Lehre/ws02/pfp/vor02WSpfp.shtml`.
- We are designing a combined, modular Eden/GpH implementation.

- We have compared the implementation designs of GpH and PMLS `http://www.macs.hw.ac.uk/~dsg/gph/papers/drafts/ifl03.ps`.

GpH Applications We have outlined a methodology for developing parallel programs in an architecture independent fashion `http://www.macs.hw.ac.uk/~trinder/papers/cameraready.pdf`.

Language We have constructed efficient implementations of several algorithmic skeletons in GdH (section 3.2.3), showing that they are more efficient, but less flexible than evaluation strategies.

Further reading:

GpH Home Page `http://www.macs.hw.ac.uk/~dsg/gph/`
Mailing list <gph@macs.hw.ac.uk>
`http://www.macs.hw.ac.uk/~dsg/gph/papers/abstracts/strategies.html`

3.2.3 GdH – Glasgow Distributed Haskell

Report by: *Henry Nyström*

The Team: *Phil Trinder, Hans-Wolfgang Loidl, Jan Henry Nystrm, Robert Pointon, Andre Rauber du Bois.*

Status: Steaming Ahead!

Implementation: An alpha-release of the GdH implementation is available on request <gph@macs.hw.ac.uk>. It shares substantial components of the GUM implementation of GpH (Glasgow parallel Haskell; section 3.2.2).

GdH Applications and Evaluation

- An EPSRC project *High Level Techniques for Distributed Telecommunications Software* (`http://www.macs.hw.ac.uk/~dsg/telecoms/`, GR/R88137) is now underway and is entering its first GdH phase. The project evaluates GdH and Erlang in a telecommunications context, the work is collaboration between Heriot-Watt University and Motorola UK Research Labs.
- GdH has been used in a MSc project at Heriot-Watt to construct efficient implementations of algorithmic skeletons for use in parallel GdH programs (see `http://www.macs.hw.ac.uk/~dsg/gdh/#papers`).
- There is a forthcoming Ph.D. thesis on the design, implementation and use of GdH by Robert Pointon (`http://www.macs.hw.ac.uk/~rpointon/`).
- GdH and Eden (section 3.2.4) are being compared, based on a distributed file server constructed in both.

Further reading:

`http://www.macs.hw.ac.uk/~dsg/gdh/`

3.2.4 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:

Madrid:	*Ricardo Peña, Yolanda Ortega-Mallén, Alberto de la Encina, Mercedes Hidalgo, Rafael Martínez, Clara Segura*
Marburg:	*Rita Loogen, Jost Berthold, Steffen Priebe*

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.

Eden has been implemented by modifying the parallel runtime system GUM of GpH (section 3.2.2). Differences include stepping back from a global heap to a set of local heaps to reduce system message traffic and to avoid global garbage collection. The current (freely available) implementation is based on GHC 5.02.3. A source code version is available via the ghc CVS repository with tag `eden-5-02-3`. We are eager to catch up to the current ghc version.

Recent Publications

skeleton-based automatic parallelisation:
Kevin Hammond, Jost Berthold and Rita Loogen: *Automatic Skeletons in Template Haskell*, 2nd International Workshop on High-level Parallel Programming and Applications (HLPP) 2003. Paris, France, June 2003.

layered structure of the Eden runtime system:
Jost Berthold, Ulrike Klusik, Rita Loogen, Steffen Priebe and Nils Weskamp: *High-level Process Control in Eden*, Euro-Par 2003 Parallel Processing, Klagenfurt, Austria, August 2003, LNCS 2790, Springer 2003, 732–741.

runtime-system level communication optimisation:
Jost Berthold: *Effects of Message Passing Mechanisms in Eden*, Draft Proceedings of Implementation of Functional Languages, IFL 2003, Edinburgh (UK), September 2003.

Eden-Maple interface:

R. Martínez and R. Peña: *Building an Interface Between Eden and Maple: Towards an Easy Parallelization of Computer Algebra Algorithms*, Draft Proceedings of Implementation of Functional Languages, IFL 2003, Edinburgh (UK), September 2003, 223–238.

non-determinism analyses:

Clara Segura and Ricardo Peña: *Correctness of Non-determinism Analyses in a Parallel Functional Language*, Draft Proceedings of Implementation of Functional Languages, IFL 2003, Edinburgh (UK), September 2003.

continuation-based semantics:

M. Hidalgo-Herrero and Y. Ortega-Mallén: *Continuation Semantics for Parallel Haskell Dialects*, APLAS'03 First Asian Symposium on Programming Languages and Systems, Beijing, China, November 27–29, Springer LNCS.

Current Activities

- Yolanda and Mercedes continue their work on the denotational semantics for Eden which is based on a continuation-based model for process creation and single-value communication. In particular, they want to extend the model to deal with streams and non-determinism.
- Rafael and Ricardo do experiments with computation-intensive computer-algebra algorithms on the recently developed Eden-Maple interface.
- Jost has started working on a new, more general implementation of parallel Haskell dialects in a shared runtime system. Starting point is the support for Eden in GHC 6.x, but the overall target is a generic parallel platform that can support multiple high-level languages and that offers implicit control of key runtime aspects such as thread management, synchronisation and communication.
- The use of Template Haskell to improve or simplify the compilation of Eden programs will be investigated by the Marburg group. In particular, Steffen's work on the polytypic skeleton library for Eden benefits from the new meta-programming facilities.

Further reading:

`http://www.mathematik.uni-marburg.de/inf/eden`

3.3 Type System/Program Analysis

3.3.1 Chameleon/A General Type Class Framework based on Constraint Handling Rules

Report by: *Martin Sulzmann and Jeremy Wazny*
Project status: *on-going*
Participants: *Gregory J. Duck, Simon Peyton Jones, Peter J. Stuckey, Martin Sulzmann, Jeremy Wazny*

We use Constraint Handling Rules (CHRs) to describe user-programmable type class extensions. In previous work, we identified sufficient conditions on the set of CHRs under which type inference is sound and decidable, and the meaning of type classes is unambiguous.
In our latest efforts, we make use of CHRs to explore and consolidate the design space for functional dependencies, and to provide for a powerful type debugging scheme.

- Functional dependencies (FDs) are a popular and useful extension to Haskell style type classes. We gave a reformulation of functional dependencies in terms of CHRs which has the following merits:
  - CHRs give us a language in which to explain more precisely what functional dependencies are. In particular, we are able to make the so-called "improvement rules" implied by FDs explicit in terms of CHRs.
  - Based on this understanding, we provide the first concise proof that the restrictions imposed by Jones on functional dependencies (described in his ESOP'00 paper) ensure sound and decidable type inference.
  - Jones's restrictions can be very limiting. We propose "more liberal FDs" which seem to be a desirable extension. We establish some concise conditions under which liberal FDs are sound. In general, liberal FDs are undecidable. Therefore, we impose a novel termination check on CHRs. We identify sufficient conditions under which CHRs are guaranteed to terminate.

- Type debugging:

  The Chameleon programming system incorporates a type debugger. This is an interactive tool for exploring types within a program, with a view to aiding programmers debug type errors. The debugger has the following features:
  - Type inference for arbitrary locations. Queries are not restricted to top-level definitions.
  - Type conflict highlighting. Type errors are reported as conflicts involving a number of program locations. Such reports contain a complete set of possibly-incorrect locations, and avoid bias inherent in traditional inference algorithms which report a single error site.
  - Type explanation. The system can explain an erroneous type by reference to the program locations which combine to give rise to that type.
  - Full support for the Haskell type system, including type classes, and experimental extensions like functional dependencies.

  Current work involves improving the nature of error reports generated by the system as well as suggesting probable fixes for type errors. In particular, we are focused on reporting fixes for subsumption errors, which can be extremely opaque when type classes are involved.

Further reading:

http://www.comp.nus.edu.sg/~sulzmann/chr/
http://www.comp.nus.edu.sg/~sulzmann/chameleon/

3.3.2 Constraint-based Type Inferencing

Report by: *Jurriaan Hage*

Participants: *Bastiaan Heeren, Jurriaan Hage, Doaitse Swierstra*
With the generation of understandable Haskell error messages in mind we have devised a constraint based type inference method which is currently being used in the Helium compiler (section 2.5.2) developed at Universiteit Utrecht.
The **main characteristics** of the inferencer are the following.

- Our philopsophy is that no single type inferencer works best for everybody all the time. Hence, we want a tunable type inferencer adaptable to the programmer's needs without the need for him to delve into the compiler.
- We generate precise position information and preserve type synonyms in error messages.
- The programmer can choose the type inference strategy of his liking (M and W and other greedy variants, and the unbiased type graph based implementations have been implemented).
- The type graph implementation uses quite a number of heuristics to decide what is the most likely source of the error.
- A logging facility is available in Helium which has given us a large amount of correct and erroneous Haskell programs which can be used to improve our type inferencer. In the future these programs can also be used for benchmarking optimizations and many other purposes. The programs have been anonymized, but the relation between programs by the same programmer has been kept intact. Various questions can then be answered: Do our hints help? Are they used? It is easy to come up with many interesting questions. Currently we have about 300 MB of sources from a single functional programming course.
- A major innovation is the ability for a programmer to develop his domain specific type rules for a combinator library he might be writing. In addition, he may specify that his experiences are that certain functions are often mixed up. As a result, a compiler may give the hint that `(++)` should be used instead of `(:)`, because `(++)` happens to fit in the context.

  The domain specific type inference rules are automatically checked for soundness, and a programmer does not have to be familiar with the process of type inferencing as it currently takes place within the compiler.

  An article on this facility can be found in the ICFP '03 proceedings (section 1.3).

Since the report of May 2003

- support for Haskell 98 type classes is now available.
- support for kind inferencing is now available. It has been added by simple reusing the machinery for types. Generating good kind error messages is still work in progress.

More future work

- we plan to develop type inference directives that apply especially to type classes. For instance, a directive that says that `Bool` will never be in the `Num` class and which then helps the compiler to decide where the error lies.
- lending support for the specification of type inference directives
- abstraction/modularization and reuse in type inference directives (type inference directives tend to be rather verbose while their are quite a few commonalities between rules for different expressions).

A new project is to extend our constraint based type inferencer to include rank-n types. This is work done with Andres Loeh and seems to work quite well. No work has been done on type error messages here.

Further reading:

Project website:
http://www.cs.uu.nl/research/projects/top/

3.4 Generic Programming

Report by: *Johan Jeuring*
Software development often consists of designing a (set of mutually recursive) datatype(s), to which functionality is added. Some functionality is datatype specific, other functionality is defined on almost all datatypes, and only depends on the type structure of the datatype.
Examples of generic (or polytypic) functionality defined on almost all datatypes are the functions that can be derived in Haskell using the deriving construct, storing a value in a database, editing a value, comparing two values for equality, pretty-printing a value, etc. Another kind of generic function is a function that traverses its argument, and only performs an action at a small part of its argument. A function that works on many datatypes is called a generic function.
There are at least two approaches to generic programming: use a preprocessor to generate instances of generic functions on some given datatypes, or extend a programming language with the possibility to define generic functions.

Preprocessors DrIFT is a preprocessor which generates instances of generic functions. It is used in Strafunski (section 4.3.3) to generate a framework for generic programming on terms.

Languages *Light-weight generic programming*: Generic functions for data type traversals can (almost) be written in Haskell itself, as shown by Ralf Lämmel and Simon Peyton Jones in *'Scrap your boilerplate'* (`http://research.microsoft.com/Users/simonpj/papers/hmap/`). The 'Scrap your boilerplate' approach to generic programming in Haskell has been further elaborated. All information is now available at `http://www.cs.vu.nl/boilerplate/` including an example suite and the documented sources. The recent extensions include functions like generic equality, `read`, and `show`. To this end, the `Data` class has been fully worked out to accommodate members for retrieving constructor and type information as well as for building terms from constructors. GHC version 6.2 (section 2.1) supports the derivation of the new extended `Data` class and comes with an accordingly extended library `Data.Generics`. A forthcoming paper by Ralf Lämmel and Simon Peyton-Jones discusses corresponding applications of scrapping even more boilerplate.

Generic Haskell: 'Dependency-style' Generic Haskell introduces a new type system for Generic Haskell that at the same time simplifies the syntax and provides greater expressive power, see the ICFP paper by Andres Löh, Dave Clarke and Johan Jeuring for a description. A type-checker has been implemented for dependency-style Generic Haskell.

Generic Haskell is used in *'Scripting XML with Generic Haskell'* by Frank Atanassow, Dave Clarke and Johan Jeuring (Proceedings SBLP'03) to implement generic XML tools, and to implement a Haskell-XML data binding from XML Schemas to Haskell. Atanassow and Jeuring show how to use this data binding together with legacy code in *'Type Isomorphisms simplify XML Programming'* (submitted for publication).

A *new generation of PolyP* has seen the light of day accompanied by a paper *'Generic programming in Haskell'* by Norell and Jansson (in submission - presented at IFL'03). The new approach embeds polytypic functions in Haskell using type classes. This means that PolyLib, the library of polytypic functions, is now available as a Haskell library. Thus the separate PolyP compiler is not strictly needed anymore. (The compiler provides a more convenient syntax for definition of new polytypic functions and it automatically derives instances for regular datatypes.)

On the more theoretical side the connection with type theory has been investigated in *'Universes for Generic Programs and Proofs in Dependent Type Theory'* by Benke, Dybjer and Jansson (to appear in Nordic Journal of Computing).

Roland Backhouse and Jeremy Gibbons have started a *project on datatype-generic programming* in August 2003, see `http://web.comlab.ox.ac.uk/oucl/research/areas/ap/dgp/`. The goal of this project is, amongst others, to develop a methodology for constructing generic programs.

Current Hot Topics: Generic Haskell: further development of the type theory and the relation between XML tools and Generic Haskell. Other: the relation between generic programming and dependently typed programming; the relation between generic programming and Template Haskell (which in prototype form has been implemented in GHC, email <template-haskell@haskell.org> to gather feedback, and see section 2.1); methods for constructing generic programs.

Major Goals: A new experimental implementation of dependency-style Generic Haskell: hopefully beginning of next year. Efficient generic traversal based on type-information for premature termination (see the Strafunski project). Exploring the differences in expressive power between the lightweight approaches and the language extension(s).

Further reading:

`http://repetae.net/john/computer/haskell/DrIFT/`
`http://www.cs.chalmers.se/~patrikj/poly/`
`http://www.generic-haskell.org/`
`http://www.cs.vu.nl/Strafunski/`
`http://www.cs.vu.nl/boilerplate/`
`http://web.comlab.ox.ac.uk/oucl/research/areas/ap/dgp/`

There is a mailing list for Generic Haskell: <generic-haskell@cs.uu.nl>. See the homepage for how to join.

3.5 Syntactic Sugar

3.5.1 Recursive do notation

Report by: *Levent Erkok*
Project status: *Implemented in both Hugs and GHC*

People: *Levent Erkok, John Launchbury*

The recursive do-notation (a.k.a. the mdo-notation) is supported by all Hugs releases since February'01, and GHC versions 6.0 and newer. (In the GHC implementation, the recursive blocks can also be marked by the keyword `rec`) Both implementations are stable and actively supported.

Further reading:

`http://www.cse.ogi.edu/PacSoft/projects/rmb/`

3.5.2 Arrow Notation

Report by: *Ross Paterson*

"GHC is full." *Simon M. (before arrow notation was added)*

Arrow notation allows one to program using John Hughes's "arrows", a generalization of monads, without being constrained to a point-free style. It has been supported for some time by a preprocessor, written in Haskell 98 and generating Haskell 98. This approach is portable, but makes it difficult for users of the notation to track their errors back to their original source. Simon Peyton Jones and I have now added direct support for arrow notation to GHC; it will be part of the upcoming 6.2 release. The notation supported differs a little from earlier versions, mainly in advanced features, and the preprocessor has been updated to match. Thus GHC will be a comfortable environment for developing arrows programs,

but they will still be runnable on other Haskell implementations, via the preprocessor.
There is also an experimental arrow transformer library in the Haskell CVS repository, and also on the arrows page. The combinators in this library are designed to work with the notation, but do rely on type class extensions currently available only in GHC and Hugs. The interface is likely to evolve. Any feedback would be welcome.

Further reading:

`http://www.haskell.org/arrows/`

Chapter 4

Libraries

4.1 Packaging and Distribution

There are various related efforts to collect libraries or to make them more visible, more easily accessible. See the following, and also the Debian Users group entry (section 6.3.1).

4.1.1 Library Infrastructure Project

Report by: *Isaac Jones*

Background: The Library Infrastructure Project is an effort to provide a framework for developers to more effectively contribute their software to the Haskell community.

The Haskell Implementations come with a good set of standard libraries included, but this set is constantly growing and is maintained centrally. This model does not scale up well, and as Haskell grows in acceptance, the quality and quantity of available libraries is becoming a major issue.

It can be very difficult for an end user to manage a wide variety of dependencies between various libraries and Haskell implementations, and to build all the necessary software at the correct version numbers on their platform: there is currently no generic build system to abstract away differences between Haskell Implementations and operating systems

The Library Infrastructure Project seeks to provide some relief to this situation by building tools to assist developers, end users, and operating system distributers.

Such tools include a common build system, a packaging system which is understood by all of the Haskell Implementations, an API for querying the packaging system, and miscellaneous utilities, both for programmers and end users, for managing Haskell software.

Status: The project is still in its infancy. A tiny prototype was implemented, along with some of the basic APIs. Consensus is gathering, however, and a document describing in detail what we intend to build is available on the project web page.

Further reading:

`http://www.haskell.org/libraryInfrastructure/`
`http://www.haskell.org/libraryInfrastructure/proposal/`

4.1.2 RPM Packaging of Haskell projects

Report by: *Jens Petersen*

RPM packages are commonly used in many Linux distributions these days. While this "project" hasn't really be formally announced, since June I have started putting RPM packages of various pieces of Haskell software that I use or find interesting up on `http://haskell.org/~petersen/rpms/`, and announced their release on the haskell list. Starting with packages of ghc, there are by now RPMs of gtk2hs, hat, hmake, greencard, HSX11, HSHGL, HaXmL, c2hs, gtk+hs, wxHaskell, hircules and darcs available. Unfortunately they are not all up to date, but if the package you want isn't you can download the older source RPM package and it should be easy to update it to the latest version: contributed rpms are very welcome. (You can of course also try poking me to get the package updated too.) It would be nice to add a apt-get/yum infrastructure, and a real webpage for the project. Also I'm considering contributing some of the major packages like ghc to the Fedora Extras project (formerly Fedora Linux project) to promote the wider acceptance and use of Haskell.

4.1.3 Haskell User-Submitted Libraries

Report by: *Shae Erisson*

The haskell-libs project is a code repository (a kind of cvs-wiki) where Haskell users can store their Haskell code and coordinate development, including documentation, releases, and bug tracking. The purpose of this repository is to centralize resources for both developers and users of Haskell software.

This sourceforge project is open for any haskell developers who wish to contribute: `http://sf.net/projects/haskell-libs/`

We also have an experimental darcs repository (section 6.1.4) on `http://www.ScannedInAvian.org/repos/hlibs/`

Some current projects that are available there:

lambdabot - an IRC bot

hws-wp - the hws webserver extended with plugins

brainfuck - a brainfuck interpreter

wiwiwi - a blog/wiki written in Haskell

pdflib - a binding to PDFLib Lite

4.2 Hierarchical Libraries

Report by: *Malcolm Wallace*

Apart from actually writing libraries that do stuff, recent meta-activity in the libraries community has concentrated on plans for making libraries easier to install, to distribute, to maintain, and therefore also easier to write and contribute to Haskellers at large.

Isaac Jones <ijones@syntaxpolice.org> has been leading the effort in bringing together a proposal for a common distribution/installation mechanism (section 4.1.1). The aim is for a general framework that can be re-used by anyone wanting to contribute a library, so that they can be sure it will work uniformly across all the compilers/interpreters without too much effort. It covers such areas as automated configure/build, library registration, dependencies on other library packages, and automatic re-installation when a compiler version is updated.

Meanwhile, the two Simons at GHC-HQ have been proposing an improved 'package' mechanism for the compilers/interpreters themselves. This proposal aims to create a more flexible 'backend' for library usage, such that libraries can be re-located within the hierarchy, and different versions of a library can co-exist together.

`http://www.haskell.org/~simonmar/packages.html`

In other recent news, the common 'base' package of libraries has recently been subdivided to separate off non-core functionality, making the core more stable, whilst enabling independent evolution of the other libraries. The split-off packages include parsec, QuickCheck, and the monad transformers. Other packages resident in the haskell.org CVS tree include OpenGL, GLUT, HaXml, Japi (Java API), ObjectIO, Win32, X11, HGL (Haskell Graphics Library), arrows, haskell-src, network, readline, and unix.

Further reading:

`http://www.haskell.org/~simonmar/libraries/libraries.html`
`http://www.haskell.org/ghc/docs/latest/html/libraries.html`
`http://www.haskell.org/mailman/listinfo/libraries/`

4.2.1 A redesigned IO library

Report by: *Simon Marlow*
Contributors:
Ben Rudiak-Gould <benrg@dark.darkweb.com>, Simon Marlow <simonmar@microsoft.com> and others.

There has been an effort underway on the libraries@haskell.org list to design a replacement for the IO library. The main aims are:

- To separate underlying IO objects (files, pipes, sockets etc.), from a general notion of Streams, providing improved
  1. Type Safety: certain operations only make sense for certain kinds of IO objects. For example `hFileSize` only makes sense on files, not sockets. Also, input streams would be separate from output streams.
  2. Generality: Under this scheme, programmers would be able to implement their own Streams (something which cannot be done with Handles).
- To allow translations to be layered on top of Streams in a general way. The most common type of translation is a text encoding, which translates between the external encoded form of text (say, UTF-8) and Haskell's Unicode Char type. This addresses a serious defficiency in Haskell's current IO library, namely the lack of support for specifying a character translation.
- More features: eg. mapped file support.

See the libraries archives for the discussion, eg.
`http://haskell.org/pipermail/libraries/2003-July/001298.html`
`http://haskell.org/pipermail/libraries/2003-July/001299.html`
`http://haskell.org/pipermail/libraries/2003-August/001313.html`

I (Simon M.) have been hacking a little on a prototype, but don't have anything significant working yet.

4.2.2 System.Process: a platform-independent API for external process control

Report by: *Simon Marlow*

A proposal for a `System.Process` library was posted to the libraries@haskell.org list:
`http://haskell.org/pipermail/libraries/2003-May/000958.html`
HTML documentation is here:
`http://www.haskell.org/~simonmar/System.Process.html`
This library is currently awaiting implementation, although I wouldn't be surprised if there were also some changes to the proposed API before it is unleashed (eg. the `createPipeEx` function looks a bit strange).

4.2.3 System.Time: a redesigned Time library

Report by: *Simon Marlow*

There has been much discussion about a replacement for the current `Time` library, because of certain problems with the existing library:

- the lack of support for leap seconds and the consequent inaccuracy of `ClockTime`
- the underspecified behaviour of `TimeDiff` / `diffClockTimes` / `addToClockTime`

The latest proposal was posted to the libraries@haskell.org mailing list, and can be found in the archives here:
`http://haskell.org/pipermail/libraries/2003-July/001290.html`
to get up to date on the discussion, be sure to read the threads which lead up to this. The majority of the discussion took place in June 2003:
`http://haskell.org/pipermail/libraries/2003-June/thread.html`
Currently, the discussion has stalled again. The leap seconds issue is something of a sticking point, and there are some implementability question marks over other parts of the API. Contribution to (any aspect of) the discussion is welcomed.

4.3 Data and Control Structures

4.3.1 The Haskell Cryptographic Library

Report by: *Dominic Steinitz*

Since this is the first report for the library, it contains information on both the first and second releases. The current release is 0.3.3.

This library collects together existing Haskell cryptographic functions and augments them so that they:

- Have common type signatures.
- Can be used with the standard mode and padding algorithms (in the case of block mode ciphers).

The library now supports: DES, Blowfish, Cipher Block Chaining (CBC) mode, PKCS5 and nulls padding, SHA-1, RSA and OAEP. The latter two based on the work done by David Sankel. In particular, you can generate key pairs using your favourite method (openssl, for example) and then use them in Haskell to encrypt and have other software decrypt and vice versa.

The library follows the hierarchical standards and has Haddock style documentation. There are demo / test programs using published test vectors and instructions on how to use RSA in Haskell and inter-work with openssl. Getting the keys into Haskell is still too manual and one of the many planned enhancements is to support PKCS12 although it may be easier to support PKCS8 first and use openssl to convert to / from PKCS12.

There is still plenty of existing code that should be incorporated such as RC4 (courtesy of Doug Hoyte).

Since it is based on existing code, it currently only works with GHC 5.04.1 or above.

Further reading:

`http://www.haskell.org/crypto/ReadMe.html`

4.3.2 HSQL

Report by: *Krasimir Angelov*

The HSQL is a simple library for database access from Haskell. Currently supported backends are ODBC, MySQL and PostgreSQL. For each backend the HSQL provides a thin FFI wrapper which allows us to connect/disconnect to a specific server and to execute SQL queries. The wrappers are designed following the same pattern, but the user can see that there are some differences which are consequences from the differences in the backend libraries. The library is written as part of my work in SmartPro Ltd but it is also free software and is available at HToolkit home page.

Further reading:

`http://htoolkit.sourceforge.net`

4.3.3 Strafunski

Report by: *Ralf Lämmel*
Project status: *no change, active, maintained*
Portability: *Hugs, GHC, DrIFT*

Strafunski is a Haskell-based bundle for generic programming with functional strategies, that is, generic functions that can traverse into terms of any type while mixing type-specific and uniform behaviour. This style is particularly useful in the implementation of program analyses and transformations.

Strafunski bundles the following components:

- the library StrategyLib for generic traversal and others;
- precompilation support for user datatypes based on DrIFT (section 3.4);
- the library ATermLib for data exchange;
- the tool Sdf2Haskell for external parser integration.

The Strafunski-style of generic programming can be seen as a lightweight variant of generic programming (section 3.4) because no language extension is involved, but generic functionality simply relies on a few overloaded combinators that are derived per datatype.

A recent application of Strafunski's StrategyLib is HaRE — the Haskell refactoring tool from Kent (section 5.3.2). It uses a good amount of functional strategies to perform transformations and helper analyses on Haskell programs.

Strafunski is currently moving to sourceforge.net. The aspiration is to simplify the distribution and to make use of the boilerplate extensions of GHC (section 2.1) as another model of functional strategies.

Further reading:

`http://www.cs.vu.nl/Strafunski/`

4.3.4 Parsec

Report by: *Daan Leijen*

Parsec is a practical parser combinator library for Haskell that is well documented, has extensive libraries, and good error messages. It is currently part of the standard Haskell libraries (in `Text.ParserCombinators.Parsec`) and has been stable for a while now. We plan to add a module that adds combinators to parse according to the (full) Haskell layout rule (available on request).

Further reading:

`http://www.cs.uu.nl/~daan/parsec.html`

4.3.5 UPC – Utrecht Parser Combinators

Report by: *Doaitse Swierstra*
(Doaitse Swierstra, Arthur Baars, Rui Guerra)

The current version of the parser combinators constructs an online result, in the sense that parts of the result can be accessed even when parsing has not yet finished. This is especially useful when parsing and processing large files of similar information. Furthermore error messages are displayed while parsing (using unsafePerformIO). The underlying mechanism for achieving this is relatively costly, although parsing speed is not much slower than that of parsers generated off line using Frown or Happy (section 5.2.1). We plan to construct a companion module (based on an earlier approach) that contains a more strict result, and which we expect to be running even faster. Furthermore the module structure may be changed by making it possible for the user of the library to tune the internals of the machine even more using classes. In order to make the everyday use of the combinators not suffer from these changes we have separated the interface and the extensions from the basic implementation, so future changes can relatively easily be made.

Furthermore three special modules were constructed, since they contain far more complex combinators, that may probably not be used by most people. Two of the combinators enable the construction of a parser that reorders the elements it has recognized (merging or permutation parsing) and keep track of this reordering by returning a function that can be used to reconstruct the original order. Inspiration for this came from the wish to be able to record the original input in such a way that error messages can be easily added to it. The third module can be used to construct parsers for languages that follow the Haskell off side rule when parsing. This turned out to be quite complicated since the precise parsing rules have been defined in terms of parse errors, and our combinators have a standard way of handling such errors; as a consequence we had to afflict some brain-damage.

Further reading:

`http://www.cs.uu.nl/groups/ST/Software/UU_Parsing/`

4.3.6 Yampa

Report by: *John Peterson*

Yampa is the culmination of the Yale Haskell Group's efforts to provide domain-specific embedded languages for the programming of hybrid systems. Yampa differs from previous FRP based system in that it makes a strict distinction between signals (time-varying values) and functions on signals. This greatly reduces the chance of introducing space and time leaks into reactive, time-varying systems. Another difference is that Yampa is structured using the arrow combinators. Among other benefits, this allows Yampa code to be written employing the syntactic sugar for arrows.

We have released a preliminary version of Yampa that contains:

- The **Yampa Base Library**, containing generic functions for the expression of signal functions operating on continuous as well as discrete signals, and advanced switching constructs for the interaction between the continuous and discrete worlds.
- The **Yampa Robotics Library**, containing entities tailored for controlling mobile robots, both real and simulated, in the style of Frob, our FRP-based robotics language. The simulator is written using Yampa's Base and HGL, the Haskell Graphics Library (section 4.6.1), and performs physical modelling of mobile differential-drive robots equipped with several kinds of sensors. A preconfigured version of the simulator allows one to play RoboCup Soccer.
- A tutorial (from the *2002 Summer School on Advanced Functional Programming*, Oxford, UK).

With the Base Library and HGL (or any other graphics library), it is easy to write reactive animation programs in the style of Fran. Thus there is no need for a special library to support graphics and animation.

In the near-term future, we aim at releasing a slightly updated version of Yampa with better documentation and more examples, including the Space Invaders game from the Haskell 2003 workshop, some minor new features to support these examples, and support for Functional Automatic Differentiation and Dirac Impulses.

Further reading:

`http://www.haskell.org/yampa`

4.3.7 The revamped monad transformer library

Report by: *Iavor Diatchki (<diatchki@cse.ogi.edu>)*

Monads are very common in Haskell programs and yet every time one needs a monad, it has to be defined from scratch. This is boring, error prone and unnecessary. Many people have their own libraries of monads, and it would be nice to have a common one that can be shared by everyone. Some time ago, Andy Gill wrote the monad transformer library that has been distributed with most Haskell implementations, but he has moved on to other jobs, so the library was left on its own. I wrote a simillar library (before I knew of the existance of Andy's library) and so i thought i should combine the two. The "new" monadic library is not really new, it is mostly reorganization and cleaning up of the old library. It has been separated from the "base" library so that it can be updated on its own. It is available from the Haskell CVS (fptools/libraries/monads). It is mostly documented with haddock (section 5.3.4). Besides reorganizing the transformers, the main changes include a new experimental transformer for non-determinism, renaming of some functions, and some new functionality here and there. There is also experimental support for resumptions, and continuations, but their interaction with the other transformers is not quite clear at the moment. If there are questions please contact me.

Further reading:

`http://cvs.haskell.org/cgi-bin/cvsweb.cgi/fptools/libraries/monads/`

4.3.8 DData

Report by: *Daan Leijen*

DData is a library of efficient data structures and algorithms for Haskell (Set, Bag, and Map). It is actively maintained and stable. We plan to add the library to the hierarchical module name space (i.e. `Data.DData.Set`) in the near future.

Further reading:

`http://www.cs.uu.nl/~daan/ddata.html`

4.3.9 HBase

Report by: *Ashley Yakeley (<ashley@semantic.org>)*

HBase is a large collection of library code, compiled "`-fno-implicit-prelude`", intended as an experimental/alternative reorganised interface to the existing standard libraries making full use of GHC's extensions. HBase development is driven by HScheme (section 6.1.1) and my other Haskell projects, and sometimes by whatever interests occur to me. Right now it includes:

- a library of various classes of Functors and Monads,
- transformation, encoding and property functions for Unicode,
- types and classes for parsing,
- functions for parsing XML and RDF,
- code for constructing SQL queries,

...and much else. I'm hoping some of the ideas might eventually make their way into standard libraries, or perhaps the standard libraries of some future extended "Haskell 2".

Further reading:

`http://sourceforge.net/projects/hbase/`

4.4 FFI

4.4.1 Template Greencard

Report: *Alastair Reid <alastair@reid-consulting-uk.ltd.uk>*
Project status: *Experimental/unstable*
Last release: *0.1 (15 Sept 2003)*
Hierarchical libraries: *No*
Portability: *GHC only (requires Template Haskell), Unix*

Template Greencard is an experimental reimplementation of Greencard (section 5.1.2) using Template Haskell.
TG is still very much under development. At present, its advantages are:

- It is very much smaller than Greencard (or any other ffi tool we know of) which should make it easier to extend and maintain than other tools.
- It is a library not a preprocessor which makes it more flexible than using a preprocessor.
- Even this early version is quite powerful.

On the downside, Template Greencard isn't as easy to use as Greencard: the syntax is worse and error messages are worse. We are not currently recommending that you stop using Greencard but, if you want to play with it a bit, feel free to look around.

Further reading:

`http://www.reid-consulting-uk.ltd.uk/projects/tg.html`

4.5 Graphical User Interfaces

4.5.1 The Common GUI Library Task Force

Report by: *Axel Simon*

The development of Haskell applications with a graphical user interface has long been complicated by the large number of mostly incomplete libraries. In spring of this year people tried to focus the development effort and came up with the idea of a Common GUI API (CGA for short) which should define an intersection of three major platform APIs (Win32, Gnome/Gtk and Mac OS X) and that addresses the requirements of the platform's style guide (or human interface guidelines). The process of defining this CGA is a major undertaking. The alternative is to use a readily available cross-platform API, like wxWindows, at the expense of creating applications that might violate a platform's style guide and cannot make use of platform specific functionality. A quick poll at the Haskell workshop revealed that 1/3 of the people thought that the CGA approach is worthwhile, 2/3 thought that the cross-platform approach is adequate. Hence the CGA idea as it stands right now will not be pursued.
The discussions at the Haskell Implementor Meeting (HIM) yielded some interesting options (or compromises) that could help to target and unify the development effort of the different library maintainers. In particular we would like to

- focus development on wxWindows (in form of wxHaskell; section 4.5.4) as this cross-platform toolkit should be adequate for most needs;
- move the programming model of Yahu (Attribute/set/get functionality) into the Haskell library hierarchy so that other (GUI) libraries can use it;
- exploit wxWindow's capability to extract platforms specific handles/objects. The goal is to use other libraries (like gtk+hs, section 4.5.5; gtk2hs, section 4.5.6; Mocca, etc.) if wxWindows does not provide the needed functionality.
- mention the names of the maintainers of each GUI library on the Haskell web site so that programmers can estimate how well-maintained a library is. Update these names regularly.

In case people feel inclined to still pursue the CGA idea, one of the following two routes could be taken:

- re-expose parts of wxHaskell in a way that forces the user to write applications that look and feel native on different platforms, i.e. a "Common GUI API on top of wxWindows."

- write a GUI Builder application for Haskell; it would surely make Haskell itself more attractive as a rapid development language (such a Builder could also cater for the look-and-feel requirements of the different platforms).

Please send any comments or critics to <gui@haskell.org>. All input (and development effort) is welcome.

Further reading:

`http://www.haskell.org/mailman/listinfo/gui/`

4.5.2 HTk

Report by: *Christoph Lüth and George Russell*
Project status: *no changes, actively maintained*

HTk is an encapsulation of the graphical user interface toolkit and library Tcl/Tk for the functional programming language Haskell. It allows the creation of high-quality graphical user interfaces within Haskell in a typed, abstract, portable and fully concurrent manner. HTk is known to run under Linux, Solaris, Windows 98, Windows 2k, and will probably run under many other POSIX systems as well. It works with GHC, versions 5.02.3 and later.

Further reading:

`http://www.informatik.uni-bremen.de/htk`

4.5.3 HToolKit

Report by: *Krasimir Angelov*

The HToolkit is a platform independent package for Graphical User Interface. The package is split into two libraries GIO and Port. The Port is a low-level Haskell 98+FFI compatible API, while GIO is a high-level user friendly interface to Port. The primary goal of HToolkit is to provide a native look and feel for each target platform. The currently supported platforms are Windows and Linux/GNOME. The 1.2 version of HToolkit supports most of the standard GUI controls and events existing in both Windows and GNOME. The current plan is to provide a full set of controls in versions up to 2.0. Some special kind of windows like: Wizards, Property sheets, Dockable Windows and simple HTML browser (based on IE explorer and GtkHTML) are also planned. Special attention will be paid to the data based controls. These are a special kind of controls which are usually not included in the standard set but are very useful in business oriented applications. An abstract document interface is planed for version 2.0. It will provide a framework to Open/Save/Edit files in accordance with the native look and feel for each platform.

Further reading:

`http://htoolkit.sourceforge.net`

4.5.4 wxHaskell

Report by: *Daan Leijen*
Project status: *beta, and actively developed.*

wxHaskell is a portable GUI library for Haskell. The goal of the project is to provide an industrial strength portable GUI library, but without the burden of developing (and maintaining) one ourselves.

wxHaskell is therefore build on top of wxWindows – a comprehensive `C++` library that is portable across all major GUI platforms; including GTK, Windows, X11, and MacOS X. Furthermore, it is a mature library (in development since 1992) that supports a wide range of widgets with native look-and-feel, and it has a very active community (ranked among the top 25 most active projects on sourceforge). Many other languages have chosen wxWindows to write complex graphical user interfaces, including wxEiffel, wxPython, wxRuby, and wxPerl.

Since most of the interface is automatically generated from the wxEiffel binding, the latest release of wxHaskell already supports about 85% of the wxWindows functionality – 2875 methods in 513 classes with 1347 constant definitions. wxHaskell has been build with GHC 6.0/6.01 on Windows, MacOS X and Unix systems with GTK. A binary distribution is available for Windows and MacOS X.

The library has improved a lot since the last community report, and supports many more features, including tree controls, toolbars, splitter windows, audio playback, openGL windows, and extensive ODBC database support.

Further reading:

You can read more about wxHaskell at `http://wxhaskell.sourceforge.net` and on the wxHaskell mailing list at `http://sourceforge.net/mail/?group_id=73133`.

4.5.5 Gtk+HS

Report by: *Manuel Chakravarty*
Project status: *beta release*

There has been no further development of Gtk+HS since the last HC&A Report. More details on the current version as well as source and binaries packages are at
`http://www.cse.unsw.edu.au/~chak/haskell/gtk/`

4.5.6 Gtk2hs

Report by: *Axel Simon*
Project status: *beta*

Gtk2hs is a wrapper around the latest Gtk release (Version 2.2 or Gtk 2 for short). Reasons for using this binding instead of, say, wxHaskell is the support of Gtk 2 specific features and the fact that gtk2hs does automatic memory management. At the moment the API is a low level veneer like Gtk+HS, but I hope to generate a Yaho/Ports-like API on top of the low level functions. It would be interesting to see if widget handles can be exchanged with wxHaskell so that applications can make use of the advanced features of Gtk 2 if they run

on the Gnome platform. Recently Duncan Coutts has added a binding to the Gtk sourceview library.
Our current work is done in a CVS repository which can be found on `http://sourceforge.net/projects/gtk2hs/`.

4.6 Graphics

4.6.1 HGL Graphics Library

Report: *Alastair Reid <alastair@reid-consulting-uk.ltd.uk>*
Project status: *Maintained, stable*
Last release: *3.00 (6 June 2003)*
Hierarchical libraries: *Yes*
Portability: *GHC, Hugs, Linux, FreeBSD, Solaris, MacOS X, Windows*
The HGL provides an easy to use, portable interface to Win32 and X11 which supports simple 2-dimensional graphics, keyboard, mouse and timer input events and multiple windows. The library is distributed as open source and is suitable for use in teaching and in applications.
The library works on both Win32 and X11 under Hugs and GHC. The API is stable and the library is used throughout Paul Hudak's 'School of Expression' textbook (`http://haskell.org/soe/`). The last release was 2.0.4 in December 2001.

Further reading:

`http://haskell.org/graphics/`

4.6.2 HSX11

Report: *Alastair Reid <alastair@reid-consulting-uk.ltd.uk>*
Project status: *Maintained, stable*
Last release: *1.00 (6 June 2003)*
Hierarchical libraries: *Yes*
Portability: *GHC, Hugs, Linux, FreeBSD, Solaris, MacOS X*
The Xlib library is a set of bindings to over 300 functions in the standard Xlib C library.

Further reading:

`http://www.reid-consulting-uk.ltd.uk/projects/HSX11.html`

4.6.3 PanTHeon

Report by: *Sean Seefried*
PanTHeon is re-implementation of Pan, a DSL embedded in Haskell, for the generation of two dimensional images and animations. (Pan was originally developed by Conal Elliott, Oege de Moor and Sigbjorn Finne.)
However this implementation differs from the former in that it is cross-platform and implemented in an entirely new way through Template Haskell (section 2.1). It is built on top of the Simple DirectMedia Layer (SDL) `http://libsdl.org` and has been successfully built on Mac OS X and Linux so far.
Instead of embedding a compiler as the original authors did (mainly for reasons of efficiency), we have opted to use the compile-time meta-programming facilities of Template Haskell to perform domain specific optimisations. This has resulted in an implementation that rivals the speed of the original, while being implemented in far fewer lines of code.

What is its status? It's almost ready for release. A paper about it can be found at `http://www.cse.unsw.edu.au/~sseefried/papers.html`

Can others get it? Unfortunately, it is not quite ready for release yet, due to problem with the current Template Haskell implementation that will be fixed in the next iteration. There are also a few features of the original library that require implementation. It retains the interactive nature of the original, unlike other re-implementations, such as Pancito, `http://www.acooke.org/jara/pancito/`.

What are the immediate plans? I plan to continue working on the interface to give it equivalent functionality to the original. Once the new version of Template Haskell comes out I will then fix the remaining problem with it which are detailed in the paper I have written on it.

4.6.4 HOpenGL – A Haskell Binding for OpenGL and GLUT

Report by: *Sven Panne*
Project status: *active, maintained*
The goal of this project is to provide a binding for the OpenGL rendering library which utilizes the special features of Haskell, like strong typing, type classes, modules, etc., but is still in the spirit of the official API specification. This enables the easy use of the vast amount of existing literature and rendering techniques for OpenGL while retaining the advantages of Haskell over lower-level languages like C. Portability in spite of the diversity of Haskell systems and OpenGL versions is another goal.
HOpenGL includes the simple GLUT UI, which is good to get you started and for some small to medium-sized projects, but HOpenGL doesn't rival the GUI task force efforts in any way. Smooth interoperation with GUIs like gtk+hs on the other hand is a goal.
Currently there are two major incarnations of HOpenGL, differing in their distribution mechanisms and APIs: The old one (latest version 1.05 from 09/09/03) is distributed as a separate tar ball and needs GreenCard plus a few language extensions. Apart from small bug fixes, there is no further development for this binding. Active development of the new incarnation happens in the fptools repository, so it is easy to ship GHC, Hugs, and nhc98 with OpenGL/GLUT support. The new binding features:

- Pure Haskell98 + FFI
- No GreenCard dependency anymore
- Full OpenGL 1.4 support (texturing and NURBS not finished yet)

- Some ARB extensions
- An improved API, centered around OpenGL's notion of state variables
- Extensive hyperlinked online documentation

HOpenGL is extensively tested on x86 Linux and Windows, and reportedly runs on Solaris, FreeBSD, OpenBSD, and Mac OS X.
Sven Eric Panitz has written a tutorial using the new API: `http://www.tfh-berlin.de/~panitz/hopengl/`

Further reading:

`http://www.haskell.org/HOpenGL/`

4.6.5 FunWorlds – Functional Programming and Virtual Worlds

Report by: *Claus Reinke*
Project status: *stalled, basic snapshot available*
FunWorlds is an "ongoing" experiment to investigate language design issues at the borderlines between concurrent systems, animated reactive 2&3d graphics, and functional programming. The only progress over the last six months (sic) has been the basic snapshot of the old system I promised in the last edition. I still hope to get back to this before Christmas, but that's what I thought last month.. so, please stay tuned, but don't hold your breath.

Further reading:

`http://www.cs.kent.ac.uk/~cr3/FunWorlds/`

4.7 Tool Frameworks

4.7.1 Medina – Metrics for Haskell

Report by: *Chris Ryder*
The Medina library is a Haskell library for GHC that provides tools and abstractions with which to build software metrics for Haskell programs.
The library includes a parser and several abstract representations of the parse trees and some visualisation systems including pretty printers, HTML generation and callgraph browsing. The library has some integration with CVS to allow temporal operations such as measuring a metric value over time. This is linked with some simple visualisation mechanisms to allow exploring such temporal data. These visualisation systems will be expanded in the near future.
We have carried out case studies to provide some validation of metrics by looking at the change history of a program and how various metric values evolve in relation to those changes. In order to do this we implemented several metrics using the library, which has given some valuable ideas for improvements to the library.
We are currently in the process of improving the visualisation systems and implementing some of the ideas from the case studies, as well as looking at ways to integrate this work with the work of the Kent FP Refactoring group.

Further reading:

`http://www.cs.kent.ac.uk/~cr24/medina/`

4.8 XML and Web Programming

4.8.1 HaXml

Report by: *Malcolm Wallace*
Project status: *stable, maintained*
HaXml provides many facilities for using XML from Haskell. The public release is due to be updated within the next few months to version 1.10, incorporating one major bugfix to the DtdToHaskell tool, a couple of minor new facilities, and re-establishing support for Hugs. Ongoing maintenance is to the CVS tree at haskell.org.

Further reading:

`http://www.haskell.org/HaXml`

4.8.2 Haskell XML Toolbox

Report by: *Uwe Schmidt (uwe@fh-wedel.de)*
Project status: *third major release*
The Haskell XML Toolbox is a collection of tools for processing XML with Haskell. It is itself purely written in Haskell. 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).
The Haskell XML Toolbox bases on the ideas of HaXml and HXML, but introduces a more general approach for processing XML with Haskell. The Haskell XML Toolbox uses a generic data model for representing XML documents, including the DTD subset and the document subset, in Haskell. This data model makes it possible to use filter functions as a uniform design of XML processing applications. The whole XML parser including the validator parts was implemented using this design. Libraries with filters and combinators are provided for processing the generic data model.
Features:

- validating XML parser
- very liberal HTML parser
- XPath support
- full Unicode support
- support for XML namespaces
- uniform data model for DTDs and XML content
- native Haskell support of HTTP1.1 and FILE protocol
- HTTP and access via other protocols via external program curl
- tested with W3C XML validation suite

- example programs

Current Work:

- XSLT implementation

Further reading:

The Haskell XML Toolbox Webpage `http://www.fh-wedel.de/~si/HXmlToolbox/index.html` includes downloads, online documentation and a master thesis describing the design of the toolbox. The design of the XPath module is described in a diploma thesis (in german).

4.8.3 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 script in one program
- a monadic, type-safe interface to generating HTML 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

New/completed Items are:

- journal paper about WASH (to appear in ACM TOIT; most up-to-date source of information)
- preprocessor for translating markup in XML syntax into WASH/HTML

Current work includes

- caching of documents
- database interface
- package-ifycation of WASH
- authentication interface

Items still on the to do list

- user manual

Further reading:

WASH Webpage `http://www.informatik.uni-freiburg.de/~thiemann/WASH/` includes examples, a tutorial, papers about the implementation.

Chapter 5

Tools

5.1 Foreign Function Interface

5.1.1 C–>Haskell

Report by: *Manuel Chakravarty*
Project status: *beta release*
C–>Haskell is an interface generator that simplifies the development of Haskell bindings to C libraries. It has recently been ported to Mac OS X and been adapted to GHC's development version 6.3. The tool is currently at version 0.12.0 and has been stress tested in the development of the Gtk+HS GUI library (section 4.5.5). Source and binary packages as well as a reference manual are available from
`http://www.cse.unsw.edu.au/~chak/haskell/c2hs/`

5.1.2 GreenCard

Report: *Alastair Reid <alastair@reid-consulting-uk.ltd.uk>*
Project status: *Maintained, stable*
Last release: *3.01 (6 June 2003)*
Hierarchical libraries: *Yes*
Portability: *Hugs, GHC, NHC and C, C++*
GreenCard is a foreign function interface preprocessor for Haskell and has been used (amongst other things) for the Win32 and X11 bindings used by Hugs and GHC.

Further reading:

`http://haskell.org/greencard/`

5.1.3 JVM Bridge

Report by: *Ashley Yakeley (<ashley@semantic.org>)*
JVM-Bridge is a GHC package intended to allow full access to the Java Virtual Machine from Haskell, as a simple way of providing a wide range of imperative functionality. Its big advantage over earlier attempts at this is that it includes a straightforward way of creating Java classes at run-time that have Haskell methods (using DefineClass and the Java Class File Format). It also features reconciliation of thread models without requiring GPH.

Current Status: A beta-quality 0.2.1 was released in July 2003 in source-code form. It compiles on Linux and Mac OS X with Sun's JVM, and should work with a number of others. A 0.2.2 release will be forthcoming shortly, allowing easier porting to Windows and better interfacing with third-party Java libraries. Further plans (0.3, or maybe 1.0) include using the hierarchical module structure, separating out pure (non-FFI) Haskell into a separate package, allowing more general assembly of Java byte-code, and making the IO-based interface cleaner and friendlier.

Further reading:

`http://sourceforge.net/projects/jvm-bridge/`

5.2 Meta Programming

5.2.1 Scanning, Parsing, and Analysis

See also constraint-based program analysis (section 3.3), the parser-combinator libraries Parsec (section 4.3.4) and UPC (section 4.3.5), and research at Utrecht (section 6.5.5).

Alex version 2
Report by: *Simon Marlow*
Project status: *version 2.0 released on August 13, 2003*
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.
Alex 2.0 is a partial rewrite of Chris Dornan's Alex 1.x, with the aim of getting it to the point of being able to generate a lexer for Haskell, and to improve the size and efficiency of the generated code. GHC itself is now using an Alex-generated lexer. Compared to the previous version of Alex, there are many changes, which are listed here: `http://www.haskell.org/alex/doc/html/about.html#RELNOTES`.

Further reading:

Alex homepage: `http://www.haskell.org/alex/`

Happy
Report by: *Simon Marlow*
Project status: *stable, maintained*
There have been no new releases of Happy since June 2002. Happy is still in constant use by GHC and other projects, and remains in maintenance mode.
Happy's web page is at `http://www.haskell.org/happy/`

The Utrecht attribute grammar system UAG

Report by: *Doaitse Swierstra*
(Arthur Baars, Doaitse Swierstra)

The Attribute Grammar system was initially developed by Doaitse Swierstra in 1999. The current version is maintained by Arthur Baars. The system reads a set of files containing an attribute grammar, in which semantic functions are described through Haskell expressions. Out of this description catamorphisms and data type definitions are generated.

The system has been bootstrapped, and now provides extensive error messages in case the attribute grammar contains errors. Only the type checking of the semantic functions is postponed to the Haskell compiler that is processing the output of the system. In a newer version we have added the conventional data flow analyses, so we may point at circularities, and can do experiments with generating more strict evaluators, of which we hope they will run even faster. The system is used in the course on Implementation of Programming Languages.

Further reading:

`http://www.cs.uu.nl/groups/ST/twiki/bin/view/Center/AttributeGrammarSystem`

5.2.2 Haskell Transformations

MAG

Report by: *Ganesh Sittampalam*
Project status: *actively maintained*

MAG is a transformation tool for a small Haskell-like language which tries to make it easy to mechanise some complex program transformations with the help of user-supplied rewrite rules. Examples this has been tried with include cat-elimination and alpha-beta pruning.

Although not much time is being spent on it, it is still actively maintained and some work is continuing on improving the higher-order matching algorithms that underpin its operation. As always, we would be very happy to hear from anyone who is interested in using it (for example for teaching), or improving it. A web interface has recently been made available so that you can try it out online - visit MAG's homepage for the link to it.

Further reading:

`http://web.comlab.ox.ac.uk/oucl/research/areas/progtools/mag/`

HsOpt: Helium/LVM Optimization in Stratego

Report by: *Eelco Visser*

HsOpt is an optimizer for the Helium compiler implemented in the transformation language Stratego. Helium is a subset of Haskell developed at Utrecht University (section 2.5.2). The optimizer works on the code produced by the Helium front-end, which is code for Daan Leijen's Lazy Virtual Machine (LVM). The goal of this project is to validate the paradigm of transformation strategies for the implementation of an optimizing compiler.

Alan van Dam has written the first version of the optimizer consisting of a basic simplifier in the style of the GHC. The main target of this simplifier has been the optimization of pattern matching code. The naive translation of pattern matching by the Helium front-end keeps it simple, but produces rather ugly code. Using a small set of transformation rules and an appropriate strategy the code can be reduced to more sane code, often similar to code that would be written by hand. This first simplification step produces an optimization of about 15%. This work is described in Alan's master thesis (`http://www.stratego-language.org/Stratego/SimplifyingTheSimplifier`).

We are planning further work on the optimizer, which would include an inliner (currently only local let bindings are inlined, not global function definitions), and to incorporate the earlier work on deforestation.

Further reading:

`http://www.stratego-language.org/Stratego/HsOpt`

Ultra

Report by: *Walter Guttmann*
status: *currently sleeping, works but should be rewritten*

Ultra is a GUI-based, semi-automatic program transformation system. The intended use is as an assistant to derive correct and efficient programs from high-level descriptive or operational specifications. The object language is an extended subset of Haskell, e.g., it does not support modules or classes, but has several descriptive (non-operational) constructs such as "forall", "exists", "some", and "that". The transformation calculus of Ultra has its roots in the Munich CIP system. Transformation rules can be combined by tactics.

What needs to be done? Well, Ultra is written in Gofer and uses TkGofer for its GUI. This means that, before any further development is going to happen, it will have to be ported to, or even completely rewritten in, Haskell. We suspect that, before that is going to happen, a "standard" GUI-library will have to emerge. It would be nice, if the new version supported complete Haskell as its object language. The semantics of Haskell is, however, quite involved compared to that of the λ-calculus, making this an ambitious project.

Further reading:

`http://www.informatik.uni-ulm.de/pm/projekte/ultra/`

5.2.3 Haskell Frontends

The Programatica Project

Report by: *Thomas Hallgren*

One of the goals of the Programatica Project is to develop tool support for high-assurance programming in Haskell.

The tools we have developed so far are implemented in Haskell, and they have a lot in common with a Haskell compiler front-end. The code has the potential to be reusable in various contexts outside the Programatica project. For example, it has already been used in the Haskell refactoring project at the University of Kent (section 5.3.2).

Further reading:

The Programatica Project, overview & papers: `http://www.cse.ogi.edu/PacSoft/projects/programatica/`
Quick overview of the tools, from the demo at the 2003 Haskell Workshop: `http://www.cse.ogi.edu/~hallgren/Programatica/HW2003/`
Executable formal specification of the Haskell 98 Module System: `http://www.cse.ogi.edu/~diatchki/hsmod/`
A Lexer for Haskell in Haskell:
`http://www.cse.ogi.edu/~hallgren/Talks/LHiH/`
More information about the tools, source code, downloads, etc: `http://www.cse.ogi.edu/~hallgren/Programatica/`

5.3 Program Development

5.3.1 Tracing and Debugging

Report by: *Olaf Chitil*

There exist a number of tools with rather different approaches to tracing Haskell programs for the purpose of debugging and program comprehension.

At the Haskell Workshop 2003, Rob Ennals presented the new Haskell debugger **HsDebug**. HsDebug provides gdb-like debugging, that is, it is used similar to traditional debuggers for imperative languages. HsDebug is based on optimistic evaluation, that is, a program is mostly evaluated eagerly, with some lazy evaluation as a fallback to preserve the non-strict semantics of Haskell. Eager evaluation ensures that the debugger shows mostly values, not unevaluated thunks, and that the nesting of stack frames relates to the call structure of the program. HsDebug is currently in a separate branch of GHC in CVS.

Hood and its variant **GHood**, for graphical display and animation, enable the user to observe data structures at given program points. Hood and GHood are easy to use, because they are based on a small portable library. They have remained unchanged for over two years.

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 with several tools in various ways: Hood-style observation of top-level functions; backwards exploration of a computation, starting from (part of) a faulty output or an error message. All tools inter-operate and use a similar command syntax. A tutorial explains how to generate traces, how to explore them, and how they help to debug Haskell programs. Hat can be used both with nhc98 and ghc. Hat can be used for Haskell 98 programs that use some language extensions.

Since the relase of Hat 2.02 on 26 March 2003, numerous bugfixes have been applied and several features added to Hat. The CVS version incorporates these. In particular, hat-detect, a tool for algorithmic debugging of Haskell programs which had been part of previous relases of Hat, has been resurrected. Hat-detect inter-operates with the existing viewing tools. Additionally, Hat now supports more of the standard hierarchical libraries. We intend to release Hat 2.04 within the next months. Also a number of further viewing tools that one day may be included in the Hat distribution are currently under development.

buddha

Report by: *Bernie Pope*
Project status: *active, version 1.0 released*

Buddha is a declarative debugger for Haskell 98. It is based on program transformation. Each module in the program undergoes a transformation to produce a new module (as Haskell source). The transformed modules are compiled and linked with a library for the interface, and the resulting program is executed. The transformation is crafted such that execution of the transformed program constitutes evaluation of the original (untransformed) program, plus construction of a semantics for that evaluation. The semantics that it produces is a "computation tree" with nodes that correspond to function applications.

Currently buddha works with GHC version 5 and 6. No changes to the compiler are needed. There are no plans to port it to other Haskell implementations, though there are no significant reasons why this could not be done.

Version 1.0 is freely available as source and is licensed under the GPL, It supports full Haskell 98 including the standard libraries. For future versions of buddha I will consider supporting common Haskell extensions. Mostly this will be driven by user requests.

Further reading:

`http://www.haskell.org/libraries/#tracing`
`http://www.cs.mu.oz.au/~bjpop/buddha`

5.3.2 HaRe – The Haskell Refactorer

Report by: *Claus Reinke*
Team: *Huiqing Li, Claus Reinke, 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.

We have just started the second year of a three-year project to explore the prospects for *'Refactoring Functional Programs'*, taking Haskell as a concrete case-study.

The last six months have been rather busy, so there's a lot to report on: In May, Simon gave an invited presentation at the 7th Brazilian Symposium on Programming Languages, accompanied by an extended abstract on *'A Case Study in Refactoring Functional Programs'*. In August, we gave a presentation about our project at the ACM Sigplan Haskell Workshop 2003 in Uppsala, Sweden, accompanied by a paper on *'Tool Support for Refactoring Functional Programs'*. Slides and papers from both events are available from our project home page.

But perhaps the most exciting development is the availability of our prototype Haskell Refactorer, **HaRe**. In May, the Programatica team (section 5.2.3) convinced their sponsors to permit a BSD-style license for their Haskell-in-Haskell frontend (thanks!), and we gave small demos of HaRe in both our presentations. Following the Haskell workshop, we started to put the first snapshots of HaRe on our website, and after things had stabilised a bit, we announced the October 1st snapshot as HaRe 0.1. It runs on a variety of platforms, comes with interface code for Emacs and Vim, and already supports about a dozen refactorings, but is not yet module- or type-aware. We continue to update the snapshot occasionally, to fix bugs you report. The main development goal for the next release is to make the existing refactorings module-aware.

Further reading:

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

5.3.3 VS Haskell

Report by: *Simon Marlow*

A project has been started to develop a Visual Studio plugin to support Haskell, with the aim of providing all the usual language-specific development environment features (eg. syntax colouring, context-sensitive help, indication of parse errors while you type), and eventually providing some more advanced features (type checking while you edit, inspecting types of identifiers or subexpressions, refactoring, debugging, GUI tools, etc.).

So far we have basic syntax coloring and parse errors in the edit buffer working, and some basic support for projects.

Help is welcome! You first need to register for the Microsoft VSIP (Visual Studio Integration Program) to get access to the VSIP SDK, which has tools, APIs and documentation for extending Visual Studio. Registering for VSIP is free, but you have to agree to a longish license agreement:
`http://www.vsipdev.com/`

If you've registered for VSIP and would like to contribute to Visual Haskell, please drop me a note (Simon Marlow <simonmar@microsoft.com>).

5.3.4 Documentation

Haddock

Report by: *Simon Marlow*
Status: *stable, maintained. Version 0.5 released July 2003*

Haddock is relatively stable, and I intend to keep maintaining it for the forseeable future. I don't have much time for wholesale improvements, although contributions are of course always welcome.

Support for implicit parameters and zip comprehensions was contributed recently by Sigbjorn Finne, and will be in the next release.

There is a TODO list of outstanding bugs and missing features, which can be found here: `http://cvs.haskell.org/cgi-bin/cvsweb.cgi/fptools/haddock/TODO`

Haddock's home page is here:
`http://www.haskell.org/haddock/`

Chapter 6

Applications, Groups, and Individuals

6.1 Non-Commercial Applications

6.1.1 HScheme

Report by: *Ashley Yakeley (<ashley@semantic.org>)*
HScheme is a project to create a Scheme interpreter written in Haskell. There's a stand-alone interpreter program, or you can attach the library to your program to provide "Scheme services". It's very flexible and general with types, and you can pick the "monad" and "location" types to provide such things as a purely functional Scheme, or a continuation-passing Scheme (that allows call-with-current-continuation) etc.

Current Status: There's an online interpreter that I keep up to date. There are a couple of major issues that stand before R5RS compliance, after which I'll make a release. See `http://hscheme.sourceforge.net/issues.php`.

Further reading:
`http://hscheme.sourceforge.net/`

6.1.2 Analysis Tools for Rosetta

Report by: *Perry Alexander*
The Systems Level Design Group at the University of Kansas Information and Telecommunication Technology Center is using Haskell to develop analysis tools for the Rosetta (`http://www.sldl.org`) requirements modeling language. We have been building upon work on modular interpreters to provide an ability to compose simulators for Rosetta domains, with the goal of performing dynamic analysis of heterogeneous models. While the work is in its early stages, we have been encouraged by preliminary results. Haskell is also being used extensively in the development of a toolset for the static analysis of Rosetta models.

Further reading:
`http://www.ittc.ku.edu/Projects/SLDG/`

6.1.3 Hircules, an IRC client

Report by: *Jens Petersen*
In the previous issue I pre-announced the first release of Hircules, an gtk2hs-based IRC client. After the 0.1 release in mid May, 0.2 was released at the start of June and 0.3 at the beginning of October. The project started as a quick hack of lambdabot (the #haskell channel IRC bot), but the code has already diverged quite a bit since then for better or worse (it would actually be really nice to factor out a nice clean IRC library from lambdabot and hircules to share in the future). Since last writing a number of features and lots of bugs have been fixed. 0.2 featured proper displaying of CTCP actions and mode changes, addition of a "/me" command, channel logging, and alerting of messages directed at the user. In version 0.3, the user interface has changed to use the channel textbuffers for both output and input; UTF-8 is supported by default, iso2022-jp messages are auto-decoded and it is possible to set the channel coding with "/coding"; also there is now tab highlighting when there is channel activity. The major features I would like to add in coming versions are auto-reconnection, support for connections to multiple servers, and a startup configuration file. Also I have plans for improvements in UI - using colours for nicks and channel highlighting, history support and more.

Further reading:
`http://haskell.org/hircules/`

6.1.4 Darcs—David's Advanced Revision Control System

Report by: *David Roundy*
Darcs is a distributed revision control system (i.e. CVS replacement), 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.
Darcs is in the process of being stabilized for a 1.0 release. It is very useable, but still has some rough edges to be worked out. A start has been made on a graphical interface using wxHaskell, although it isn't expected to be stabilized for the 1.0 release. Darcs is free software licensed under the GNU GPL.
Further reading:
`http://www.abridgegame.org/darcs`

6.1.5 Yarrow, a proof-assistant for Pure Type Systems

Report by: *Frank Rosemeier*

From the Yarrow home page:
"A proof-assistant is a computer program with which a user can construct completely formal mathematical proofs in some kind of logical system. In contrast to a theorem prover, a proof-assistant cannot find proofs on its own.
Yarrow is a proof-assistant for Pure Type Systems (PTSs) with several extensions. A PTS is a particular kind of logical system, defined in 'Henk Barendregt. Lambda Calculi with Types. In Handbook of Logic in Computer Science, vol. 1. Oxford University Press, 1992'. In Yarrow you can experiment with various pure type systems, representing different logics and programming languages. A basic knowledge of Pure Type Systems and the Curry-Howard-de Bruijn isomorphism is required. (This isomorphism says how you can interpret types as propositions.) Experience with similar proof-assistants can be useful."
Frank Rosemeier reports that he has ported Yarrow (written by Jan Zwanenburg, in Haskell 1.3, see `http://www.cs.kun.nl/~janz/yarrow/`), to Haskell 98.
The Haskell 98 source code will be published on his homepage this year, probably in November: `http://www.fernuni-hagen.de/MATHEMATIK/ALGGEO/Mitarbeiter/Rosemeier/rosemeierengl.htm`.

6.2 Commercial Applications

6.2.1 Reid Consulting Ltd

Report: *Alastair Reid <alastair@reid-consulting-uk.ltd.uk>*

Reid Consulting Ltd. offers Haskell consulting, contracting and training. Involved in Haskell since its early development, we have played a key role in turning Haskell into a language that you can use to build successful products.
Services we provide:

- Supporting open-source compilers, tools and libraries
- Developing libraries
- Creating Haskell bindings to non-Haskell libraries
- Performance tuning
- Code reviews
- Training new staff
- Training in advanced Haskell techniques

Using our services will increase your rate of success, reduce your development time and help you develop a better product.

Further reading:

`http://www.haskell-consulting.com/`

6.2.2 Aetion Technologies LLC

Report: *Mark T.B. Carroll (<Mark.Carroll@Aetion.com>)*
Aetion Technologies LLC is a small American defense contractor that uses Haskell and Java for most of its software development. The larger current Haskell-based projects we are working on involve (a) automated reasoning under uncertainty, currently focusing on the interpretation of sensor data, and (b) an object-oriented modeling language for composable simulations. Additionally, we develop small Haskell programs for a variety of tasks from document processing to time tracking.
A number of our projects involve systems of entities that react to changes in each other, so we are currently investigating Functional Reactive Programming as an appropriate framework for implementing such systems. We are looking at Haskell-based declarative GUI toolkits and Web Authoring System Haskell (WASH) as a useful basis for some future work that will involve more user interaction. A project that we are about to start work on will involve implementing a server that manages a distributed computation.
In addition, Aetion donates some programmer time to community projects like the Library Infrastructure Project and the Haskell Experimental Debian Archive, mentioned elsewhere in this report (sections 4.1.1 and 6.3.1).
Further reading:
`http://www.aetion.com/`

6.2.3 Binary Parser

Report: *Sengan Baring-Gould*
<sengan.baring-gould@amd.com>
Sengan Baring-Gould at AMD is developing a binary parser which given a grammar is able to extract fields from values. This is used as part of an internal ICE (hardware debugger) and a State Extractor tool, which eases the debugging of chips by extracting a slice of a failing program from the signals on the RAM and PCI pins and makes it into a test that can be run in the CPU's simulation environment.
Binary parser provides the ability to reference by name values which may be composed of other values. It goes one step further in that the client program does not need to know where a particular value is buried, only what its value is. Binary parser grammars are intended to enable non-programmers to access fields of their registers, without requiring the tool-developer to write explicit code to do so. For instance a technical writer could write a binary parser grammar for a device of which the tool developer has never heard. Stress has been put on generality and simplicity, rather than efficiency. For instance binary parser allows multiple definitions, cyclic definitions, etc.
Binary Parser is a Haskell library linked into State Extractor which is written in x86 assembly, C, `C++`, and Python. The Python interface to Binary Parser was written using Haskell's FFI and the Boost libraries. It provides a very intuitive interface for non-functional programmers. For instance the following sets the CPU simulation environment up to start execution at the reset vector.

```
  x86.eip = 0xFFF0
  x86.cs  = 0xF000
```

Binary parser simplifies the porting of the internal tools from chip to chip where the location of register-fields may change but their functionality does not.

6.3 Haskell User Groups

6.3.1 Debian Users

Report by: *Isaac Jones*

There are many Debian users in the Haskell community, and they have recently begun an initiative to form a more coherent group. This involves serious packaging work, especially by Ian Lynagh to bring new binary versions of GHC, NHC, and other packages to various versions of Debian.

The group is working toward a solution for the longstanding problems with binary distribution of Haskell packages, with discussion taking place on the Haskell Wiki (`http://www.haskell.org/hawiki/DebianUsers`). It is hoped that the Library Infrastructure Project (section 4.1.1) will help here.

In order to provide backports, bleeding edge versions of Haskell tools, and a place for experimentation with packaging ideas, Isaac Jones has started the "Haskell Experimental" Debian archive (`http://www.syntaxpolice.org/haskell-experimental/haskell-experimental.html`) where a wide variety of packages can be found.

6.4 Haskell in Education

6.4.1 Beseme Project

Report by: *Rex Page*

The Beseme Project seeks to provide ideas and materials for covering the standard material of a one-semester course on discrete mathematics for computer science and engineering students.

A distinctive element of the Beseme approach is that discrete mathematics concepts are illustrated with examples from software development, rather than the usual examples from number theory, graph theory, and the like. This gives computing students an opportunity to see applications of the theory in a context that interests and motivates them, without sacrificing any of the usual mathematical concepts.

Statistics gathered over a four-semester period and analyzed using standard methods based on Student's t-distribution suggest that the Beseme approach gives students a leg up in a subsequent course on data structures that has a heavy programming component. Specifically, students with above-average, overall grade-point averages who took the Beseme course earned higher marks than above-average students who took a course with similar mathematical content, but illustrated with traditional examples.

According the the t-statistic model, there is only a 2% likelihood that the difference (between the average grade in the data structures course of the Beseme students and that of the traditional students) can be explained by random effects, and other analyses suggest that aspects such as quality of instruction and intellectual abilities of the students also do not explain the difference, leaving course content as a likely, influential factor.

The analysis is discussed in greater detail in a paper that appeared in ICFP 2003, entitled *"Software Is Discrete Mathematics"*. The paper, along with other reports on the Beseme Project, is accessible through the Beseme website.

Course materials available on the website include lectures notes (in both PowerPoint and PDF form, over 350 slides in all), homework (about 100 problemns and solutions), examinations (about 200 questions and solutions), and a syllabus and lesson plan.

About two-thirds of the material of the course centers around mathematical logic. After the introduction of predicates, all of the examples in the logic portion of the course involve reasoning about properties of software, most of which is expressed in Haskell (a few are conventional looping functions). Software examples include sum, sequence concatenation, logical operations on sequences, the Russian peasant algorithm, insertion and lookup in AVL trees, and other computations. Most of the properties verified relate to aspects of program correctness, but resource utilization properties are also verified in some cases.

The remaining third of the course discusses other standard topics in discrete mathematics, such as sets, functions, relations, trees, and counting.

The Beseme website provides access to a preview of the material. Exams and solutions are protected by a login procedure (to increase the comfort level of instructors wishing to use them in courses). To locate the website, just google "Beseme".

6.5 Research Groups

6.5.1 Artificial Intelligence and Software Technology at JWG-University Frankfurt

Report by: *Matthias Mann, David Sabel*
Members: *Matthias Mann, David Sabel, Manfred Schmidt-Schauß*

DIAMOND A current research topic within our DIAMOND project is understanding side effects and Input/Output in lazy functional programming languages using non-deterministic constructs.

We introduced the **FUNDIO** calculus which proposes a non-standard way to combine lazy functional languages with I/O. FUNDIO is a lazy functional core language, where the syntax of FUNDIO has `case`, `letrec`, constructors and an IO-interface: its operational semantics is described by small-step reductions. A contextual approximation and equivalence depending on the Input/Output behavior of normal order reduction sequences have been defined and a context lemma has been proved. This enables us to study a semantics and semantic properties of the language. By using the technique of complete reduction diagrams we have shown a considerable set of program transformations to be correct. Several optimizations of evaluation are given, including strictness optimizations and an abstract machine, and shown to be correct w.r.t. contextual equivalence. Thus this calculus has a potential to integrate non-strict functional programming with a non-deterministic approach to Input/Output and also to provide a useful semantics for this combination.

We applied these results to Haskell by translating the GHC core language to the FUNDIO language. Based on an extended set of correct program transformations for FUNDIO, we investigated the local program transformations, which are performed in GHC. The result is that most of the transformations are correct w.r.t. FUNDIO, i.e. retain sharing and do not force the execution of IO-operations that are not needed. By turning off the few transformations which are not FUNDIO-correct and those that have not yet been investigated (especially the global ones), we have achieved a FUNDIO-compatible modification of GHC called **HasFuse**.

HasFuse compiles Haskell programs which make use of `unsafePerformIO` in arbitrary contexts. Since the call-by-need operational semantics of FUNDIO considers all IO-operations as independent unless they are made sequential using data dependency, the behaviour of `unsafePerformIO` is no longer 'unsafe'. This means, the user does not have to undertake the proof obligation that the timing of an IO-operation wrapped by 'unsafePerfomIO' does not matter in relation to all the other IO-operations of the program. So `unsafePerformIO` may be combined with monadic IO in Haskell, and since all the reductions and transformations are correct w.r.t. to the FUNDIO-semantics, the result is reliable in the sense that no IO-operation will unexpectedly be duplicated.

Ongoing work is, beside others, devoted to the proof of correctness of further program transformations.

Further reading:

Chair for Artificial Intelligence and Software Technology
`http://www.ki.informatik.uni-frankfurt.de`
DIAMOND `http://www.ki.informatik.uni-frankfurt.de/research/diamond`

6.5.2 Formal Methods at Bremen University

Report by: *Christoph Lüth and Christian Maeder*

Members: *Christoph Lüth, Klaus Lüttich, Christian Maeder, Achim Mahnke, Till Mossakowski, George Russell, Lutz Schröder*

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

The **UniForM** workbench is a tool integration framework mainly geared towards tools for formal methods. It is actively used and developed further in the MMiSS project. The workbench currently contains over 80k lines of Haskell code (plus a few hundred lines of C).

We are further using Haskell to develop tools, like parsers and static analysers, for languages from the **CASL** family, in particular CASL itself, **HasCASL**, and **HetCASL**, which combines several specification languages such as CSP, CASL, HasCASL, and Modal and Temporal Logic.

We use the Glasgow Haskell Compiler (GHC), exploiting many of its extensions, in particular concurrency, multiparameter type classes, hierarchical name spaces, functional dependencies, existential and dynamic types. Further tools actively used are DriFT, Haddock, the combinator library Parsec, and the haskell-src package.

Further reading:

Group activities overview: `http://www.informatik.uni-bremen.de/agbkb/forschung/formal_methods/`
UniForM workbench
`http://www.informatik.uni-bremen.de/uniform/wb`
HTk Graphical User Interfaces for Haskell Programs
`http://www.informatik.uni-bremen.de/htk`
MMiSS Multimedia instruction in safe systems
`http://www.mmiss.de`
CASL specification language
`http://www.informatik.uni-bremen.de/cofi`
Heterogeneous tool set
`http://www.informatik.uni-bremen.de/cofi/hets`

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

Report by: *Murray Gross*

One prong of the Metis Project at Brooklyn College, City University of New York, is research on and with Parallel Haskell in a Mosix-cluster environment. At the present time, with the assistance of the developers at Heriot Watt University (Edinburgh) and elsewhere, we have implemented a PVM-free version of GUM for use under Mosix on i86 machine for release 5 of GHC, and we are currently porting this release to Solaris for use in SMP environments under Solaris. Some interesting preliminary results concerning performance under Mosix are being examined, and we hope to be able to present a technical report on the issues that have been raised sometime later this fall.

Further reading:

`http://www.sci.brooklyn.cuny.edu/~metis`
Contact: Murray Gross, <magross@its.brooklyn.cuny.edu>.

6.5.4 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 (see also section 2.5.1). In addition to current projects mentioned in previous reports:

- Qingsong Ye and a number of other students are looking at designing embedded DSLs for specifying different aspects of handheld applications, including data synchronisation and user interface.

Further reading:

Our web page is still being re-developed:
`http://www.comp.mq.edu.au/plrg/`
In the meantime, please contact us via email to <plrg@ics.mq.edu.au>.

6.5.5 Functional Programming at Utrecht University

Report by: *Daan Leijen*

All UU Software (`http://www.cs.uu.nl/groups/ST/`)
We are well on our way to make all our Haskell modules mutually consistent and to make them available through a CVS server at `cvs.cs.uu.nl`, in the directory `uust`. Currently included are our parser combinators, pretty printers and attribute grammar system. Further software will be added in the near future.

Parser Combinators *(Doaitse Swierstra, Arthur Baars, Rui Guerra)* See the description of UPC in section 4.3.5.

Helium *(Arjan van IJzendoorn, Bastiaan Heeren, Daan Leijen)* See the description of Helium in section 2.5.2.

Improving Type Errors *(Bastiaan heeren, Jurriaan Hage, Doaitse Swierstra)* See the description of the constrained based inferences in section 3.3.2.

The attribute grammar system AG *(Arthur Baars, Doaitse Swierstra)* See the description of UAG in section 5.2.1.

Utrecht Haskell Compiler *(Atze Dijkstra, Doaitse Swierstra)*
Project status: *active development*
formerly: *Type Checker for Extended Haskell*
The type checker we started a while ago has been growing and quickly approaches a more or less complete Haskell compiler, supporting many extensions we think might go into Haskell 2, such as higher rank types (both existential and universal types), polymorphic kinds etc. Some small programs have ran successfully on the Lazy Virtual Machine that was developed by Daan Leijen.

Pretty Printing *(Pablo Azero, Doaitse Swierstra)*
Our pretty printing combinators have been silently doing their work over the years. Currently we are updating them, so they can be generated by the new version of the AG system. They too will have a more flexible interface allowing naming of subformats by using a monadic top layer.
`http://cvs.cs.uu.nl/cgi-bin/cvsweb.cgi/uust/lib/pprint/`

Proxima *(Martijn Schrage, Johan Jeuring, Lambert Meertens, Doaitse Swierstra)*
In the proxima we are designing a layered system for building interactive editors. An interesting aspect of our approach is that we have designed combinators for glueing the different layers that take part in the online formatting. By our knowledge this is the only place where combinators are used to combine really large program structures. We are currently in the state that we can produce small editors in a relatively easy way.
Some screenshots of editors that have been written you may take a look at the projects home page at: `http://www.cs.uu.nl/groups/ST/twiki/bin/view/Center/Proxima`

Syntax Macros *(Arthur Baars, Doaitse Swierstra)*
Project status: *actively developed*
The syntax macros are now in a state that one gets a macro mechanism for free when using our attribute grammar system and parser combinators in constructing a front end of a compiler. Necessary glueing code is automatically generated. The syntax macros make it possible to extend the context free grammar of a language on a per program basis. Examples of constructs that no longer have to be part of the standard language, but could have been defined us- ing our macro mechanism are the **do**-notation, **arrow**-notation and the notation for list comprehensions. Currently we manage even to give the user feedback in terms of his original program, by allowing online redefinition of the attribute grammar that constitues the compiler.
The current version is available at `http://www.cs.uu.nl/groups/ST/twiki/bin/view/Center/SyntaxMacros`

First Class Attribute Grammars *(Arthur Baars, Doaitse Swierstra)*
We are investigating how to make language definitions more compositional, and how to capture recurring patterns of analysis and data flow in compilers. Ideally we should like to have so-called first class aspects. It is a matter of research however how to integrate type checking and aspect oriented programming. Attempts using extendible records almost seem to do the job, but unfortunately incorrect use leads to pages of error messages. We hope that by following the techniques explained in http://www.cs.uu.nl/people/arthurb/dynamic.html may help to solve the problem.

6.5.6 Functional Programming at the University of Kent

Report by: *Claus Reinke*

We are a group of about a dozen staff and students with shared interests in functional programming. While our work is not limited to Haskell, it provides a major focus and common language for teaching and research.
Our members pursue a variety of Haskell-related projects, many of which are reported in other sections of this report. *Keith Hanna* is continuing his work on visual interactive programming with Vital (see section 2.5.4). *Axel Simon* develops

the Gtk2hs binding to version 2.2 of the Gtk GUI library (section 4.5.6) and has also been trying to coordinate the Haskell GUI efforts (section 4.5.1). *Chris Ryder* has evaluated his Metrics and Visualization library Medina through some case studies, and is now working on improvements (section 4.7.1). *Huiqing Li*, *Simon Thompson* and *Claus Reinke* have released first snapshots of HaRe, the Haskell Refactorer (section 5.3.2), and Claus Reinke has had no time for his favourite project combining functional programming and virtual worlds (section 4.6.5).

Further reading:

FP group:
`http://www.cs.kent.ac.uk/research/groups/tcs/fp/`
Vital: `http://www.cs.kent.ac.uk/projects/vital/`
Gtk2HS: `http://gtk2hs.sourceforge.net/`
FunWorlds: `http://www.cs.kent.ac.uk/~cr3/FunWorlds/`
MEDINA: `http://www.cs.kent.ac.uk/~cr24/medina/`
Refactoring Functional Programs:
`http://www.cs.kent.ac.uk/projects/refactor-fp/`

6.5.7 Programming Languages & Systems at UNSW

Report by: *Manuel Chakravarty*

PLS is a young research group at the University of New South Wales whose Haskell-related activities comprise high-performance arrays for Haskell, whole program optimisation of Haskell programs, optimisation of Embedded Domain Specific Languages (EDSLs) in Haskell, Haskell to Java translation, and a Haskell to ObjectiveC bridge. We also contribute to the Glasgow Haskell Compiler. Moreover, we work on the use of λ-calculus as an intermediate language for optimising compilers of conventional languages, the safe execution of untrusted code, Python for the Single Address Space Operating System (SASOS) Mungi, and cluster computing.

Further reading:

Further details about PLS and the above mentioned activities can be found at `http://www.cse.unsw.edu.au/~pls/` or be obtained by sending an email to <pls@cse.unsw.edu.au>

6.5.8 Institute for Geoinformation at TU Vienna

Report by: *Andrew Frank*

Haskell used for Geographic Information Science Research We have used Haskell and primarily Hugs for the past 5 years as a language to formalize complex problems in Geographic Information Science:
Specification of interfaces: interoperability between programs of different vendors requires the definition of interfaces. The industry group Open GIS Consortium (`http://www.opengis.org`) is part of the international standardization process under ISO (ISO TC 211) and uses the conventional methods (UML, English text). This leads regularly to interpretation problems: what is meant with a specific interface description? What is the correct implementation? We have demonstrated that Haskell can be used to write the specifications in an unambiguous way. Haskell is executable and this produces the additional benefit that results for interesting questions can be produced automatically (Frank and Kuhn 1995; Frank and Kuhn 1998; Kuhn and Frank 1998).

Modeling cognitive spatial agents: the complex processes of observation of environment, decision making and action in space analyzed; we construct computational models in Haskell. A first model of a very simple situation (finding the way to the gate in an airport) was successful (Raubal, Egenhofer et al. 1997; Raubal and Egenhofer 1998; Raubal and Worboys 1999; Raubal 2000; Raubal 2000; Raubal and Frank 2000; Raubal 2001; Raubal 2001). Another computation model analyzes the process of making maps based on observation of an environment and then using the same maps by another agent for wayfinding (Frank 2000). The formal models allowed us to compare wayfinding in a real environment with wayfinding in the web (Hochmair 2000) (Hochmair 2000; Hochmair 2001; Hochmair and Frank 2001; Hochmair and Raubal forthcoming). Ongoing work is concentrating on users of public transportation: Ms. E. Pontikakis is integrating wayfinding with the business process of ticket buying etc.

Building computational models to understand real estate ownership and the related process in a cadastre (propriety registry). In one effort we built a computational model to John Searle's concept of 'socially constructed reality' (Searle 1995; Smith and Searle 2001) and applied it to real estate (Anderson, Birbeck et al. 2000; Bittner, Wolff et al. 2000; Steffen Bittner 2002; Bittner to appear). In a second effort, the Austrian cadastral law was translated, paragraph by paragraph, in Haskell to allow formal analysis of its content (Navratil 1998; Navratil 2002; Navratil and Frank 2003)

At the core of much of our work is the representation of collections of facts; the ordinary relational data model (Codd 1970; Codd 1982) does not integrate well with current object-oriented design paradigms and functional approaches. We explore a data model based on relations, which links to category theory (Bird and de Moor 1997). We have a running system which allows flexible storage and retrieval of multiple relations. This is reminiscent of work done in the early 80s (Shipman 1981), which did not succeed in the imperative programming environment.

Geographic Information Systems are very complex, large programs and therefore very difficult to analyze and to teach. It seems possible to reconstruct the data processing part of a GIS (not the user interface) in Haskell and identify the algebras relevant. Haskell permits the integration of different parts of mathematics (algebraic topology, projective geometry, linear algebra, etc.) in a uniform setting.

Further reading:

`http://www.geoinfo.tuwien.ac.at/research/researchtopics.htm`

6.6 Individual Haskellers

"What are you using Haskell for?" – the implementation mailing lists are full of people sending in bug reports and feature suggestions, stretching the implementations to their limits. Judging from the "reduced" examples sent in to demonstrate problems, there must be quite a few Haskell applications out there that haven't been announced anywhere (probably because Haskell is "just" a tool in those projects).

If you're one of those serious Haskell users, why not write a sentence or two about your application? We'd be particularly interested in your experience with the existing tools (e.g., that all-time-favourite: how difficult was it to tune the resource usage to your needs, after you got your application working? Which tools/libraries where useful to you? What is missing?).

Oleg Kiselyov <oleg@pobox.com> writes:

Simon Peyton Jones noted, "Haskell has become a laboratory and playground for advanced type hackery". A recently added page `http://pobox.com/~oleg/ftp/Haskell/types.html` gives examples of the type hackery, some of which seems to be actually useful. One such example is a polyvariadic composition *mcomp*, which gives the *farthest* composition of two functions. If $f :: a1 \to a2 \to .... \to cp$ and $g :: cp \to d$ then f '*mcomp*' $g :: a1 \to a2 \to .... \to d$ where *cp* is not a functional type. The page `types.html` shows one application of the polyvariadic composition, for an automatic uncurrying of deeply nested tuples. The latter lets us write categorical products in Haskell in a natural way: e.g., $max7 = max3.((max2 * max3) * max2)$.

The page `types.html` also shows how to dynamically dispatch on a *class* of a type. In other words, how to emulate $IsInstanceOf$. Finally, the page describes AVL trees with a blended static and dynamic enforcement of the tree balancing constraint. The function *make_node* verifies the constraint at compile time – if it can. If the static check is not possible, the function delays the check till the run-time.

Linked from `types.html` is a page that discusses several approaches to number-parameterized types, i.e., datatypes that depend on unary or decimal numbers such as arrays with a statically-checked size. `http://pobox.com/~oleg/ftp/Haskell/number-parameterized-types.html` The latter have been described in the previous edition of the Report.

Graham Klyne (`http://www.ninebynine.org/`, `http://www.ninebynine.net/`) writes:

My primary interest is in RDF (`http://www.w3.org/RDF/`) and Semantic Web (`http://www.w3.org/2001/sw/`) technologies, and I am a participant in the W3C RDFcore working group (`http://www.w3.org/2001/sw/RDFCore/`). I see the Web in general, and the Semantic Web in particular, as a natural territory for application of functional programming techniques. I aim to use Haskell as a "scripting language" for the Semantic Web – to develop applications based on simple inferencing over RDF data, overcoming limitations I have encountered with available off-the-shelf RDF inference tools. I find it particularly appealing that Haskell has the full power of a general purpose programming language, but supports a programming style that can be matched closely to formal and semi-formal specifications to provide extra validation for Internet/Web protocol definitions.

My current motivating application is using RDF in network configuration applications (`http://www.ninebynine.org/SWAD-E/Intro.html#HomeNetAccessDemo`), using inference rules to map general network policy descriptions (in RDF) to device-specific configuration files or instructions.

The first elements of this work, a Notation3 parser and an RDF graph comparison and merging utility, were released and announced in June 2003 (`http://www.haskell.org/pipermail/haskell/2003-June/012013.html`). I have since completed an RDF query processor that forms the heart of a simple inference and proof-checking framework, and am currently working towards implementing a framework for adding RDF datatype deductions, and packaging the whole for release and experimentation.

Future directions:

- application to network device configuration and access control
- application to trust modelling (cf. `http://www.ninebynine.org/iTrust/Intro.html`)
- a full RDF/XML parser based on the new proposed RDF syntax specification (`http://www.w3.org/TR/rdf-syntax-grammar/`).
- fully or partially automated inference/proof discovery.
- integration with RDF storage systems implemented in Java and/or C (e.g. Jena `http://www.hpl.hp.com/semweb/`)
- many code refinements as my understanding of Haskell technique grows.

Markus Schnell writes:

I'm working on a Ph.D. thesis (I'm in my third year) on Concept-to-Speech Systems. For implementation of the algorithms I use ghc under Windows 2000 and Mac OS X. Whenever possible, I want to make available some of the modules. These can be found on my webpage: `http://www.markusschnell.com/haskell.html` Due to copyright problems it's mainly a module for understanding Hidden Markov Models (HMM) and the Viterbi Algorithm.

In my spare time I'm putting together a Web Content Manager in Haskell to make maintaining my website easier. For that, I'm growing a FTP module, which should be available soon.

Steffen Mazanek is working on his MSc thesis (expected Jan 2004), *"Higher-kinded types in the context of subtyping"*, at the University of the German Federal Armed Forces, Munich.

Abstract: Type systems significantly support the modern software enineering process. Therefore a lot of effort is invested to improve the existing methods. From the object-oriented programming paradigm we have noted the benefits of extension and specialization gained by inheritance and subtyping mechanisms, i.e., enhanced productivity, reusability and hence robustness and maintainability.

The type system of Haskell is still missing an appropriate treatment of subtype relations. Starting from Nordlanders approach `O'Haskell` (`http://www.math.chalmers.se/~nordland/ohaskell/`) we introduce a new application of higher-kinded types in this context. Thereby special function kinds and kind variables are used to keep track of variances so that even complex type expressions behave as expected. This result is achieved, because we establish a subtype relation between higher-kinded types.
Our kind system motivates another extension. We present a subkind relation (in analogy to subtyping) to state, that a particular kind is more specific than another one. Amazingly, the arrow kind constructor behaves antitone in its first argument similarly to the arrow type constructor.
We aim at simplicity, speed of execution and compatibility to standard Haskell98 as well as `O'Haskell`. Furthermore we compare our approach with several well-known type system extensions and provide a basic implementation.

Further reading:

`http://www.steffen-mazanek.de/diplomarbeit.html`
Please note, that this thesis is not yet published (it will be in early January) and hence the website given above does not contain comprehensive information by now.

Ketil Z. Malde <ketil+haskell@ii.uib.no> writes:
I'm developing various tools for sequence analysis as part of my Ph.D. work in bioinformatics. Currently available is "xsact", a fast EST clustering program using a novel (in this context) algorithm based on a sort-of suffix array.
Soon to come is a consensus sequence generator. The total size is about 2500 lines of Haskell.
Everything, including code, published articles and presentation, is or will be available at `http://www.ii.uib.no/~ketil/bio/`

Tom Pledger <Tom.Pledger@peace.com> writes:
Since 2001 my work has mainly been the design and implementation a business data processing language. It is lazy, functional, and has a lot in common with discrete Functional Reactive Programming. It is impure inasmuch as it reads from a database, and loses referential transparency at the end of each transaction, when it relinquishes its locks and clears its cache. In early 2003 the project team grew to two people. Our interpreter is written in

- Haskell (98 plus the Control.Monad libraries) for the static analysis part, and
- Nice (`http://nice.sourceforge.net`) for the runtime part, because it must run in a Java Stored Procedure in an Oracle database, with no access to native code.

It's at the prototype stage. Our next steps are to test and refine it further, and to add a component for visualising calculations - the Auditors' Replay.
I take care not to ask for related free advice on the Haskell mailing lists, because my employer expects to own and profit from this product.

Lloyd Allison <lloyd@mail.csse.monash.edu.au> writes:
Inductive Inference I am using Haskell to examine what are the products[1] of AIDMIIMLSI[2] from a programming point of view, that is how do they behave, what can be done to each one, and how can two or more be combined? The primary aim is the getting of understanding, and that could one day be embodied in a useful library or prelude for AIDMIIMLSI[2].
Currently there are types and classes for models (various probability distributions), function models (including regressions), time-series (including Markov models), mixture models, and classification trees.
The URL `http://www.csse.monash.edu.au/~lloyd/tildeFP/II/` contains some references, and code will be available there if and when it is a bit more polished and published.

[1] E.g. Mixture-models (unsupervised classification, clustering), classification- (decision-) trees (supervised classification, expert systems), Bayesian/causal networks/models, etc..

[2] AIDMIIMLSI = artificial-intelligence/ data-mining/ inductive-inference/ machine-learning/ statistical-inference/ etc., call it what you will. (Super Thunder Sting Car Ray Bird – Pete & Dud.)

Alain Crémieux <alcremi@pobox.com> writes:
I work in a company making software for payrolling, which is a incredibly complex subject in France. I'd like to use Haskell in my everyday's work, but up to now it's only been a personal hobby ("passion" would be more accurate). I am convinced that embedded languages ("DSEL") have strong applications in software-making companies. Of course Haskell is one of the best language for that; but if there are many fascinating papers (resulting from hard work & deep insights) on the subject, practical reusable code lacks. Most projects are unfinished, and when code exists it is in "as is" state, and never reusable without much work. So as a start I am trying to implement *"How to write a financial contract"* `http://research.microsoft.com/Users/simonpj/Papers/financial-contracts/pj-eber.ps`, a chapter in "The Fun of Programming" by Simon Peyton Jones & Jean-Marc Eber. This article details the full analysis (the hard part) of an embedded language application. A deriving implementation should provide several solutions, depending on design decisions (for instance a "shallow" embedding and a "deep" embedding), simplifications rules leading to a (moderately) optimised interpreter and a code generator (`C--` is my target language). And the code should work (calculate the value of several common financial options). Two other key papers for this work are Compiling Embedded Languages `http://www.conal.net/papers/jfp-saig/`, by Conal Elliott, Sigbjorn Finne and Oege de Moor, and Programming Graphics Processors Functionally `http://www.conal.net/papers/Vertigo/` (Vertigo) by Conal Elliott. Any advice and/or discussion would be welcome. I have 2 other projects going on, an Haskell binding to Graphviz (rather advanced), and a Haskell binding to Berkeley DB.
