Haskell Communities and Activities Report

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

Seventh Edition – November 10, 2004

Andres Löh (ed.)

Perry Alexander, Lloyd Allison, Krasimir Angelov
Alistair Bayley, Jérémy Bobbio, Björn Bringert
Paul Callaghan, Mark Carroll, Manuel Chakravarty
Olaf Chitil, Koen Claessen, Andrew Cooke
Catarina Coquand, Duncan Coutts, Philippa Cowderoy
Alain Crémieux, Iavor Diatchki, Atze Dijkstra
Peter Diviánszky, Shae Erisson, Sander Evers
Simon Foster, Leif Frenzel, John Goerzen
Murray Gross, Walter Guttmann, Jurriaan Hage
Sven Moritz Hallberg, Thomas Hallgren, Keith Hanna
Dean Herington, Anders Höckersten, John Hughes
Graham Hutton, Patrik Jansson, Johan Jeuring
Isaac Jones, Oleg Kiselyov, Graham Klyne
Daan Leijen, Andres Löh, Rita Loogen
Salvador Lucas, Christoph Lüth, Ketil Z. Malde
Christian Maeder, Simon Marlow, Conor McBride
Serge Mechveliani, Brandon Moore, Andy Moran
Matthew Naylor, Henrik Nilsson, Jan Henry Nyström
Sven Panne, Ross Paterson, Jens Petersen
John Peterson, Simon Peyton-Jones, Jorge Sousa Pinto
Bernie Pope, Alastair Reid, Claus Reinke
Frank Rosemeier, David Roundy, George Russell
Chris Ryder, David Sabel, Uwe Schmidt
Axel Simon, Peter Simons, Anthony Sloane
Dominic Steinitz, Donald Bruce Stewart, Martin Sulzmann
Henning Thielemann, Peter Thiemann, Simon Thompson
Phil Trinder, Tuomo Valkonen, Eelco Visser
Joost Visser, Malcolm Wallace, Ashley Yakeley
Jory van Zessen

Preface

Welcome to the Seventh edition of the Haskell Communities and Activities report. I can proudly announce that the report has survived yet another change of editor, and chances are good that this won't be the last report I edit, so you are going to have to live with me for a while.

First of all, I want to emphatically thank the two previous editors, Claus Reinke and Arthur van Leeuwen, not only for the visible results they have produced (the previous six reports), but also for providing an infrastructure (template files, contact lists, little tools etc.) with which the preparation of this report has been more entertaining than stressful. In this context, I also want to thank John Peterson, who helped out promptly and unbureaucratically when I happened not to be satisfied with the software installed on haskell.org.

Furthermore, I want to thank everyone in the rather impressive list of contributors. Without your work, these reports would not be possible, and I appreciate the many friendly replies I got to my calls and reminders, and I especially thank all first-time contributors! In my – rather subjective – impression, we have a significant increase in the number of Haskell-based applications. Successful applications are certainly a good way to help Haskell to more recognition outside the core community.

Still, I have the feeling that the report could be better and more complete. From time to time, I find myself reading a mail on the mailing list mentioning a project I have never heard of, or even announcing a new project. When I ask if a contribution to the HC&A Report is forthcoming, I usually get the answer "well ... now that you asked, it is". While such a reaction is of course positive, I cannot possibly catch all new or hidden projects this way. Therefore, I rely on you, the readers, to provide feedback, and to communicate with each other: please encourage yourself and others you know are working on Haskell projects to contribute, and please tell me ⟨hcar@haskell.org⟩ about things you are missing from the Report. And please, **mark already the last weeks of April in your calendar, as the contributions to the May 2005 edition are due by then**!

I couldn't stop it and change a few TEXnical things: I sincerely hope that you like the new look of the report. The main functional modification is that you can detect changes more easily. Entries that are (nearly) unchanged with respect to the previous edition are listed normally (they will still contain current and interesting information, though). Entries that have been updated have a header with a blue background, and finally all-new entries (again, w.r.t. the previous edition) find themselves completely on a blue background.

I don't particularly like the historically grown categorization of the Report. I apologize in advance for all the projects I placed wrongly or suboptimal (the commercial *users* are listed under "Applications", for example, whereas research and user groups have their own chapter). For the next Report, I will think about modifying the overall structure a bit: suggestions are welcome.

Ok, I've said enough – read it while it's hot!

Andres Löh, University of Utrecht, The Netherlands

Contents

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.

Thanks to Fritz Ruehr for making the cafepress store on haskell.org a lot more exciting and to Jonathan Lingard for adding some nice style sheets to our pages.

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:

- 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.

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 `#haskell`

Report by: Shae Erisson

The `#haskell` IRC channel is a real-time text chat where anyone can join to discuss Haskell. Point your IRC client to irc.freenode.net and join the `#haskell` channel.

The `#haskell.se` channel is the same subject but discussion happens in Swedish. This channel tends to have a lot of members from Gothenburg.

The `#darcs` channel has been added since the last HC&A Report, if you want real-time discussion about darcs, drop by!

1.3 The Haskell HaWiki

Report by: Shae Erisson

The Haskell wikiwiki is a freely editable website designed to allow unrestricted collaboration. The address is http://www.haskell.org/hawiki/. Some highlights are:

- http://www.haskell.org/hawiki/CommonHaskellIdioms
- http://www.haskell.org/hawiki/FundamentalConcepts

Feel free to add your own content!

1.4 Books and tutorials

1.4.1 New textbook – Programming in Haskell

Report by: Graham Hutton

I am currently in the final-stages of producing an introductory Haskell textbook. The book is a revised and extended version of my Haskell course at the University of Nottingham, which has been developed and class tested over many years. The first seven chapters (97 pages) are available for preview on the web: http://www.cs.nott.ac.uk/~gmh/book.html

I'd be pleased to make the full current draft (162 pages) available to anyone that is teaching Haskell and may be interested in using the book in their course; please contact me for further details.

1.4.2 hs-manpage-howto(7hs)

Report by: Sven Moritz Hallberg

While writing the manpages for Pesco.Cmdline, I assembled some guidelines to follow with respect to structure and formatting in their own manpage, hs-manpage-howto(7hs). It's a rough document far from complete, and mainly meant as a reminder and guide for myself, but if anyone else would like to document his or her Haskell modules with roff (which I hope to encourage hereby), it might well prove useful.

Alas, if you write a Haskell manpage, and come up with a style guideline not covered, please let me know!

Further reading

http://www.scannedinavian.org/~pesco/man/html7/hs-manpage-howto.7hs.html

1.5 Haskell related events

1.5.1 Past events

CUFP

Report by: Andy Moran

Functional languages have been with us for a little over a generation now. Languages like Lisp, Scheme, ML, OCaml, Haskell, and Erlang have well engineered compilers and tools and large user and development communities. Not surprisingly, some of these users work in industry. While there are only a few companies where functional languages are used exclusively, there are many that use functional languages for design exploration, prototyping, modeling, specification and design, building compiler-like tools, and even for developing product.

In recognition of this the first Annual ACM SIGPLAN Workshop for Commercial Users of Functional Programming (CUFP) was held on September 18th, 2004, in Snowbird, Utah. It was co-located with the 2004 ACM SIGPLAN International Conference on Functional Programming (ICFP).

The goals of the workshop were to act as a voice for commercial users of functional programming languages and technology; to help functional programming become increasingly viable as a technology for use in the commercial, industrial, and government space, by providing a forum for functional programming professionals to share their experiences and ideas, whether business, management or engineering, and to enable the formation and cementing of relationships and alliances that further the commercial use of functional languages.

There were 25 attendees, and the workshop was based around 9 short presentations from a diverse cross-section of commercial users, representing Abstrax Inc., Microsoft, Linspire Inc., Beckman Coulter Inc., Cadence Research Systems, Galois Connections, Inc., and Bluespec Inc. Participants were encouraged to view the speakers as leading discussion, as opposed to giving academic presentations, and this led to spirited and fruitful discussions.

In short: the workshop was very successful, and we hope to make it a regular event.

Throughout the day, a number themes emerged from talks and the discussions they engendered. An article describing those themes in more detail is available (http://www.galois.com/cufp/CUFP-Report.pdf).

The Succ Zeroth IOHCC

Report by: Shae Erisson

The Succ Zeroth International Obfuscated Haskell Code Contest was a great success! Thanks to all those who entered, please enter again next year! See the results here: http://www.scannedinavian.org/iohcc/succzeroth-2004/.

ICFP Programming Contest 2004

Report by: Andres Löh

Haskell did extremely well in this year's seventh ICFP programming contest. The participate the contest, a given task has to be solved within 72 hours (with a special "lightning-division" prize available for the best solution received within 24 hours). There are no restrictions on team size, and any programming language can be used.

This year's task was to design an ant colony that will bring the most food particles back to its anthill, while fending off ants of another species. To win the contest, one had to submit the neural wiring for the ants in your colony – a text file containing code for a simple, finite state machine that is run by all of the ants.

Each team was allowed to submit two programs, and in the final ranking the top four entries were written by teams who used Haskell. The winning team was "Dunkosmiloolump" (Ian Lynagh, Ganesh Sittampalam, Andres Löh, Duncan Coutts). The second place

team was "The Frictionless Bananas" (Jeremy Sawicki and Mieszko Lis). In total, 230 teams participated, out of which 20 used Haskell (in comparison: 25 C++, 24 OCaml, 23 hand-coded, 21 Java, 16 Python, 15 C, 12 Lisp, 11 Pearl, 9 Scheme, ...).

Further reading

- Contest home page: http://www.cis.upenn.edu/proj/plclub/contest/
- Team "Dunkosmiloolump": http://urchin.earth.li/icfpcontest/2004/
- Team "The Frictionless Bananas": http://www.sawicki.us/icfp/2004/

AFP 2004

Report by: Andres Löh

This year in August, the 5th International Summer School on Advanced Functional Programming took place in Tartu, Estonia. About 70 people with different backgrounds gathered for a week to learn and teach several interesting topics related to functional programming. Naturally, the Haskell language had a prominent place among the lectures. The topics were:

- Programming with arrows (→ 3.5) (by John Hughes)
- Epigram (→ 2.6.7): Dependent types for practical programming (by Conor McBride)
- Combining datatypes and effects (by Alberto Pardo)
- A strongly typed functional operating system based on dynamics (by Rinus Plasmeijer)
- Generic programming (→ 3.4) techniques for the construction of graphical user interfaces (by Rinus Plasmeijer)
- Declarative debugging with Buddha (→ 5.4.3) (by Bernie Pope)
- Typing Haskell with an attribute grammar (→ 3.3.3) (by Doaitse Swierstra and Atze Dijkstra)
- Server-side web programming in WASH (→ 4.7.4) (by Peter Thiemann)
- Refactoring functional programs (→ 5.3.4) (by Simon Thompson)

Certainly, the lecturers are willing to provide further information about these topics to interested people.

One beneficial side effect of such a Summer School is that the lecturers produce lecture notes, which usually are excellent tutorials to the respective subjects. The lecture notes of this Summer School will be published as an LNCS volume. I wish to thank the local organizers Varmo Vene and Tarmo Uustalu for making this a wonderful and enlightening event.

Further reading

http://www.cs.ut.ee/afp04/

1.5.2 Future events

You may want to participate in some of the following Haskell-related events:

TFP 2004 The 5th Symposium on Trends in Functional Programming will commence in only a few weeks, on November 25 and 26, in Munich, Germany. See http://www.tcs.informatik.uni-muenchen.de/~hwloidl/TFP04/.

POPL 2005 The 32rd Annual Symposium on Principles of Programming Language, is held next year from January 12 to 14, in Long Beach, California. See http://www.cs.princeton.edu/~dpw/popl/05/.

PADL 2005 The 7th International Symposium on Practical Aspects of Declarative Languages is co-located with POPL 2005. More at http://www.unm.edu/~herme/padl05/.

TLCA 2005 The 7th International Conference on Typed Lambda Calculi and Applications will take place in Nara, Japan, on April 21–23, 2005.

PPDP 2005 The 7th International Symposium on Principles and Practive of Declarative Programming will be held in Lisboa, Portugal, July 11–13, 2005, with a CFP at http://www.site.uottawa.ca/~afelty/ppdp05/, co-located with

ICALP 2005 The 32nd International Colloquium on Automata, Languages and Programming, also in Lisboa, Portugal, July 11-15, 2005. Additional information at http://icalp05.di.fct.unl.pt/.

2 Implementations

2.1 The Glasgow Haskell Compiler

Report by: Simon Peyton-Jones

Here are some development highlights from the last few months. They will all be incorporated in GHC 6.4, which we will release before Christmas.

- The new back end, advertised in the last Communities Newsletter is complete. The back-end infrastructure is now much simplified, and can compile to `C--` (though that particular route is not yet solid). All the run-time system "`.hc`" files are now "`.cmm`" files, and are compiled by GHC itself. That in turn means that they can all be compiled via GHC's native code generator, with no use of `gcc` at all. Furthermore, pofiling, ticky-ticky stuff etc all work with the native code generator, whereas before they required `gcc`.
- Wolfgang Thaller has added initial support for position-independent code (PIC).
- Cabal (Haskell's new package management system) (→ 4.1.1) is included with, and partially supported by, GHC. There is more to do: for example, we don't yet deal with package versions, or packages that expose some but not all of their modules.
- A major new development is the inclusion of Generalised Algebraic Data Types (GADTs) in the type system. GADTs offer a pretty substantial increase in expressivenes. There's a paper about GADTs here http://research.microsoft.com/~simonpj/papers/gadt/index.htm
- When compiling mutually-recursive modules, you have to hand-write a "`.hi-boot`" file, and it's easy to get that wrong. GHC now checks for consistency, which eliminates a potent and embarassing source of segmentation faults.
- New libraries incoroprated in GHC: `System.Process`, `Network.URI` (thanks to Graham Klyne (→ 8.2)), and various DData libraries (`Map`, `IntMap`, `Set`, `IntSet`; thanks to Daan Leijen (→ 4.4.1)).

As part of the Visual Studio work (→ 5.5.3), we now plan to make GHC itself into a proper Haskell library, with a well-defined interface. So you should be able to say "`import GHC`" and then call the parser, typechecker, and so on. It'll have a pretty big interface, of course, and we'll emit draft specifications in the next few months. If you're a potential user of a GHC-as-a-library, do take a look at the drafts and let us know whether they'll work for you.

Not much progress on the Template Haskell front, largely due to lack of interest. If there's an active user community, you are keeping very quiet, and so TH has slipped down our priority list.

As ever, we are grateful to the many people who submit polite and well-characterised bug reports. We're even more grateful to folk 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.

2.2 Hugs

Report by: Ross Paterson
Status: stable, actively maintained, volunteers welcome

The most recent release of Hugs was in November 2003. The development version incorporates support for Unicode, thanks to Dimitry Golubovsky ⟨dimitry@golubovsky.org⟩, increased support for the hierarchical libraries, and numerous bug fixes. It is high time for another release, and Hugs is mostly ready, except that more work is needed on Windows. It would be great if the next release could support the Graphics library (used in Paul Hudak's book) on Windows, as it will on X11. If it works with Hugs it will probably work with GHC too, but there is no-one to do the necessary fixing.

The next release is planned to include more third party libraries than previous ones, though in such a way as to make separate upgrades of these libraries fairly painless. The idea is to provide a substantial Haskell system out of the box. Library authors who would like to participate should make their libraries work with Hugs and contact us. The Cabal project (→ 4.1.1) is also developing Hugs support.

The manpower available for Hugs development and maintenance remains very limited. Contributions from volunteers are welcome. Sven Panne has made a Windows binary available; test reports would be welcome. Even better would be people prepared to build, test and debug on Windows. (A full build requires one of the free Unix-like environments for Windows.)

Dimitry Golubovsky has also developed an introspection extension to Hugs (hugs-users, September), extending the limited experimental features already present to provide full access to Hugs's compilation results. He has in mind applications like saving for use with an alternative runtime, precompilation and more, but would like to hear from anyone who is interested.

2.3 nhc98

Report by:	Malcolm Wallace
Status:	stable, maintained

nhc98 is a small, easy to install, standards-compliant compiler for Haskell 98. It is in stable maintenance-only mode – the current public release is version 1.16, but a bug-fix refresh version 1.18 is imminent. Maintenance continues in CVS at haskell.org, and implementation hackers are invited to play with nhc98's internals if they wish.

Further reading

http://haskell.org/nhc98

2.4 Haskell-Clean Compiler

Report by:	Peter Diviánszky
Status:	experimental

About our Haskell-Clean compiler found at http://aszt.inf.elte.hu/~fun_ver/#ToC11:

- It is an experimental version.
- It could be updated because the vesion of current Clean System is 2.1 and it was developed with version 2.0.2. (However, the distribution contains the 2.0.2 Clean System.)
- We are working on a refactoring tool for Clean. We intend to refactor Haskell programs with the same tool. If it will be possible, our Haskell-Clean compiler will be revisited. Until then, we do not think we will develop or update it.

2.5 Haskell to Clean Translation

Report by:	Matthew Naylor

The primary aim of the project is to develop a tool, which we name Hacle, for translating Haskell programs to Clean programs, thereby allowing the Clean compiler to compile Haskell programs. The question is, *can the Clean compiler, in combination with Hacle, produce faster executables than existing Haskell compilers?*

The answer, perhaps rather predictably, is *sometimes yes*. We have noticed that, in some cases, the hybrid Hacle-then-Clean compilation system can produce executables which are up to a factor of four times faster than the corresponding GHC-compiled programs. However, we suspect that these cases are in a minority. Nevertheless, to be of any significance at all, we must also argue Hacle's completeness.

Hacle can translate programs which conform to a slightly restricted Haskell 98 standard. It can translate itself, which is written in approximately fifteen thousand lines of code and makes use of many of the features provided by Haskell 98. This result positively demonstrates reasonable completeness.

The project is effectively finished; this is not to say that the tool cannot be improved, rather that we are content with its current state. Only the unlikely event of widespread use would motivate such improvements. However, the following question is unanswered: why do Clean and GHC sometimes outperform each other?

For more information including detailed technical documentation, my dissertation, more results, Hacle's limitations, and a download link to Hacle, see the project's web page.

Further reading

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

Grateful acknowledgements to Malcolm Wallace and Olaf Chitil.

2.6 Variations of Haskell

2.6.1 Haskell on handheld devices

Report by:	Anthony Sloane
Status:	unreleased

Work on our port of nhc98 (→ 2.3) to Palm OS is continuing but, unfortunately, is not ready for public release at this stage.

2.6.2 Helium

Report by:	Daan Leijen
Participants:	Arjan van IJzendoorn, Bastiaan Heeren, Daan Leijen, Rijk-Jan van Haaften
Status:	stable

The purpose of the Helium project is to construct a light-weight 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 (→ 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. One of the aspects of the compiler is that can log errors to a central repository, so we can track the kind of problems students are having, and improve the error messages and hints.

Currently, the Helium compiler has been used successfully for the third time during the functional programming course at Utrecht University. There is also initial support for type classes, but we are still investigating the quality of error messages in the presence of overloading.

Further reading

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

2.6.3 Educational Domain Specific Languages

Report by:	John Peterson
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 programming 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. Version 1.0 of Pan# was released in March and the system finally has a type checker. Pan# is an excellent introduction to functional programming and can be used in conjunction with the Fun of Programming chapter as an excellent way to teach functional languages. Our website contains a number of examples produced by this language and some instructional materials.

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 March, 2005.

Further reading

http://haskell.org/edsl/

2.6.4 Vital: Visual Interactive Programming

Report by:	Keith Hanna
Status:	active (latest release: May 2004)

Vital is a highly interactive, visual environment that aims to present Haskell in a form suitable for use by engineers, mathematicians, analysts and other end users who often need a combination of the expressiveness and robustness that Haskell provides together with the ease of use of a 'live' graphical environment in which programs can be incrementally developed.

In Vital, Haskell modules are presented as 'documents' having a free-form layout and with expressions and their values displayed together. These values can be displayed either textually, graphically (as linked data structures) or pictorially, and can be edited using conventional Copy/Paste mouse gestures. This gives end users an intuitive way of inputting or modifying complex literal data structures. For example, a value of type Tree can be displayed graphically and subtrees selected, copies and pasted between nodes.

Recent development in Vital include the ability for animated and interactive displays (a release of this system is planned for November).

The present implementation includes a collection of interactive tutorial documents (including examples illustrating approaches to Exact Real Arithmetic).

The Vital system can be run via the web: a single mouse-click is all that is needed!

Further reading

Home page: http://www.cs.kent.ac.uk/projects/vital/

2.6.5 hOp

Report by:	Jérémy Bobbio and Thomas Hallgren
Status:	beta, active development

hOp is a micro-kernel based on the run-time system (RTS) of the Glasgow Haskell Compiler. It is meant to enable people to experiment with writing various components of an operating system in Haskell. This includes device drivers, data storage devices, communication protocols and tools required to make use of these components.

The February 2004 release of hOp consisted of a trimmed-down RTS that does not depend on features usually provided by an operating system. It also contains low-level support code for hardware initialization. This release made most functions from the base hierarchical library available (all but the System modules), including support for threads, communication primitives, and the foreign function interface (→ 3.1).

Building on the features of the initial release, we designed and implemented an interrupt handling model. Each interrupt handler is run in its own thread, and sends events to device drivers through a communication channel. We tested our design by implementing a simple PS/2 keyboard driver, and a "shell" that allows running a "date" command, which accesses the real time clock of the computer. A release of hOp containing these additional features was made in June 2004.

Iavor Diatchki, Thomas Hallgren, and Andrew Tolmach made some additions to hOp. The additions include a PS/2 mouse driver, using VBE 2.0 to setup a linear frame buffer for graphics, a window system implemented in Haskell (Gadgets, developed by Rob Noble and Colin Runciman at the University of York), new primitives for setting up demand paged virtual memory and executing arbitrary machine code in protected mode. The function to launch a user space executable is parameterized by a system call handler, a page fault handler and a timeout. The resulting system is in an experimental state and is preliminary called House.

Further reading

Further information, source code, demos and screenshots are available here:

- http://www.macs.hw.ac.uk/~sebc/hOp/
- http://www.cse.ogi.edu/~hallgren/House/

2.6.6 Agda: An Interactive Proof Editor

Report by:	Catarina Coquand
Status:	active development

Agda is an interactive type-based editor for editing proofs and programs that has been developed at Chalmers and Göteborg University. It builds on previous work at Chalmers such as ALF and Cayenne. It implements a proof/type checker for a language that is based on Martin-Löf Type Theory. We are experimenting with how such a proof langauge could be extended with data-types, modules and records. The syntax of the language is rather close to Haskell. The language can also be seen as a start for a dependently typed programming language.

The program is written in Haskell and it consists of roughly 15 000 lines of code. It is connected with one graphical and one text-based interface. The graphical interface Alfa http://www.cs.chalmers.se/~hallgren/Alfa/ is written in Haskell using Fudgets. The is also a "simple" emacs-interface which doesn't know the syntax of the language and communicates via a text-based protocol with Agda. This interface comes with the distribution of Agda.

Agda is running with a stable version that is slightly more than one year old. It is also possible to download newer unstable versions. In this new version experiments are done with hidden arguments as in Cayenne, addition of over-loading with a class system and built-in types such as characters, strings and integers.

We have recently started a collaboration with AIST (Advanced Industrial Science and Technology Institute in Japan) on development and applications of Agda. In particular on writing better documentation and integration with other automatic proof tools.

Agda source code can be browsed at http://cvs.coverproject.org/marcin/cgi/viewcvs/ and can be accessed by anonymous CVS from cvs.coverproject.org.

Short term goals are among many things:

- Write a better documentation of the code and the system.
- Examples of classes and built-in types
- Building on the libraries
- Revision of the type-checking algorithm
- Connecting Agsy with the emacs-interface – Agsy is an automatic proof search plugin for Alfa for the moment.

Further reading

For more details about the project, read about QuickCheck (→ 5.4.4) and Cover (→ 7.1.10) in this report or consult the homepage at http://www.cs.chalmers.se/~catarina/agda/.

2.6.7 Epigram

Report by:	Conor McBride

Epigram is a prototype dependently typed functional programming language, equipped with an interactive editing and typechecking environment. High-level Epigram source code elaborates into a dependent type the-

ory based on Zhaohui Luo's UTT. The definition of Epigram, together with its elaboration rules, may be found in 'The view from the left' by Conor McBride and James McKinna (JFP 14 (1)). The former has implemented Epigram in Haskell, interfacing with the xemacs editor.

Motivation

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

Specifically, Epigram is designed to support programming with inductive datatype families indexed by data. These provide a means to incorporate strong logical invariants which well typed programs are guaranteed to preserve. Examples include matrices indexed by their dimensions, expressions indexed by their types, search trees indexed by their bounds. Correspondingly, we can express matrix multiplication taking (Matrix i j) and (Matrix j k) to (Matrix i k), tagless evaluators, sorting algorithms which produce sorted output.

Dependent types enable us to express statically what is learned when a test is performed. In Epigram, this informative testing often takes the form of derived pattern matching principles or 'views': if you can prove that a set of expressions cover a type, then you may use those expressions as patterns. For example, any pair of natural numbers is either $(x, x + y)$ or $(y + 1 + z, y)$. This gives us a way to test if $m \leq n$, observing and exploiting what this test tells us about m and n.

Implementation

Whilst Epigram seeks to open new possibilities for the future of strongly typed functional programming, its implementation benefits considerably from the present state of the art. On the language side, considerable use is made of monad transformers, higher-kind polymorphism and type classes. Moreover, its denotational approach translates Epigram's lambda-calculus directly into Haskell's. On the tool side, Haskell's profiler (in the capable hands of Paul Callaghan) has proved invaluable for detecting bottlenecks in the code.

Current Status

Epigram can be found on the web at http://www.dur.ac.uk/CARG/epigram and its community of experimental users communicate via the mailing list ⟨epigram@durham.ac.uk⟩. The current implementation is naive in design and slow in practice, but it is adequate to exhibit small examples of Epigram's possibilities. At time of writing, a new implementation is in design, incorporating a compiler based on Edwin Brady's research. Recent successful funding bids have ensured that the development will continue.

3 Language Extensions

3.1 Foreign Function Interface

Report by:	Manuel Chakravarty
Status:	Version 1.0

Version 1.0 of the Haskell 98 FFI Addendum is available. The report has been through many revisions and is fully implemented by GHC and Hugs and mostly implemented by NHC98. As with Haskell 98, the FFI standard is meant to be a stable interface that Haskell developers can rely on in the midst of new extensions and language features. Details are available from http://www.cse.unsw.edu.au/~chak/haskell/ffi/.

What is missing, at the moment, is a good tutorial that serves as a companion to the standards document and explains FFI programming by way of a comprehensive set of examples. If anybody feels the urge to help out by contributing all or parts of such a tutorial, please let me know at ⟨chak@cse.unsw.edu.au⟩.

3.2 Non-sequential Programming

3.2.1 GpH – Glasgow Parallel Haskell

Report by:	Phil Trinder
Participants:	Phil Trinder, Abyd Al Zain, Andre Rauber du Bois, Kevin Hammond, Leonid Timochouk, Yang Yang, Jost Berthold, Murray Gross

Status

A complete, GHC-based implementation of the parallel Haskell extension GpH and of evaluation strategies is available.

System Evaluation and Enhancement

The first 3 items are linked by a British Council/DAAD collaborative project between Heriot-Watt University, St Andrews University, and Phillips Universität Marburg.

- We are adapting GpH to run on computational GRIDs. The current implementation performs well on single clusters, and multiple clusters with a low-latency interconnect. A distribution is available on request from ⟨ceeatia@macs.hw.ac.uk⟩.
- We are designing a generic parallel runtime environment encompassing both the Eden (→ 3.2.3) and GpH runtime environments
- In separate work GpH is being used as a vehicle for investigating scheduling on the GRID.
- We are teaching parallelism to undergraduates using GpH at Heriot-Watt and Phillips Universität Marburg.

GpH Applications

GpH is being used to parallelise the GAP mathematical library in an EPSRC project (GR/R91298).

Implementations

The GUM implementation of GpH is available in two development branches, and work on a port of GUM to the latest GHC 6.xx branch has been started over summer.

- The stable branch (GUM-4.06, based on GHC-4.06) is available for RedHat-based Linux machines: binary snapshot (see installation instructions). The stable branch is available from the GHC CVS repository via tag gum-4-06.
- The unstable branch (GUM-5.02, based on GHC-5.02) is working and has been used on a Beowulf cluster. It is available on request as a source bundle.

Our main hardware platform are Intel-based Beowulf clusters. Work on ports to other architectures is also moving on (and available on request). Specifically a port to a Mosix cluster has been built in the Metis project at Brooklyn College, with a first version available on request from Murray Gross.

Further reading

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

3.2.2 GdH – Glasgow Distributed Haskell & Mobile Haskell

Report by:	Jan Henry Nyström
Participants:	Phil Trinder, Hans-Wolfgang Loidl, Jan Henry Nyström, 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 (→ 3.2.1). A beta release of mHaskell will be available in December 2005.

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 a collaboration between Heriot-Watt University and Motorola UK Research Labs.

- 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/).

Further reading

- The GdH homepage:
  http://www.macs.hw.ac.uk/~dsg/gdh/
- The mHaskell homepage:
  http://www.macs.hw.ac.uk/~dubois/mhaskell

3.2.3 Eden

Report by: Rita Loogen

Description

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

in Madrid: Ricardo Peña, Yolanda Ortega-Mallén, Mercedes Hidalgo, Rafael Martínez, Clara Segura

in Marburg: Rita Loogen, Jost Berthold, Steffen Priebe, Pablo Roldán Gómez

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 (see above). 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 from the Eden web page. Installation support will be provided if required.

Recent Publications

survey and new standard reference Rita Loogen, Yolanda Ortega-Mallén and Ricardo Peña: Parallel Functional Programming in Eden, accepted for the Journal of Functional Programming special issue on Functional Approaches to High-Performance Parallel Programming 2004, to appear.

semantics M. Hidalgo-Herrero: *Formal Semantics for a parallel functional language*, Ph.D. Thesis, Universidad Complutense de Madrid, June 2004 (in Spanish).

compilation Steffen Priebe: *A Framework for Enhancing Eden Code with Template Haskell*, Proceedings of of the Workshop on Implementation of Functional Languages, IFL 2004, Lübeck, Germany, September 2004.

generalised runtime system Jost Berthold: *Towards a Generalised Runtime Environment for Parallel Haskells*, Workshop on Practical Aspects of High-level Parallel Programming (PAPP 2004), ICCS, Kraków, Poland, June 2004.

profiling Pablo Roldán Gómez: *Eden Trace Viewer: A Tool to Visualize Parallel Functional Program Executions*, Diploma Thesis, Universidad Complutense de Madrid, July 2004 (in german).

skeleton performance analysis Jost Berthold, Rita Loogen: *Analysing Dynamic Channels for Topology Skeletons in Eden*, Proceedings of the Workshop on the Implementation of Functional Languages, IFL 2004, Lübeck, Germany, September 2004.

Current Activities

- Yolanda and Mercedes continue their work on semantics for parallel functional languages, in particular Eden. Mercedes has finished her Ph.D. thesis on formal semantics for a parallel functional language.

- Jost is working on a more general implementation of parallel Haskell dialects in a shared runtime system.
- Steffen continues his work on the polytypic skeleton library for Eden making use of the new metaprogramming facilities in GHC.
- The Eden trace viewer developed by Pablo in the context of his diploma thesis provides new insights into the runtime behaviour of Eden programs.

Further reading

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

3.2.4 HCPN – Haskell-Coloured Petri Nets

Report by:	Claus Reinke
Status:	new project

Coloured Petri Nets are a high-level form of Petri Nets, in which anonymous tokens are replaced by data objects of some programming language (and transitions can operate on that data, in addition to moving it around). The combination of functional languages and Petri nets promises a rich design space – the two formalisms have little overlap and much to offer to each other.

Haskell-Coloured Petri Nets (HCPN) are an instance of this hybrid graphical/textual modelling formalism for Haskell. So far, we have a bare-bones graphical editor for HCPN, building on the portable wxHaskell GUI library (→ 4.5.2). From this, HCPN can be saved, loaded, and exported as Haskell code for graphical or textual simulation. HCPN NetEdit and NetSim are not quite ready for prime time yet, but functional; as long as you promise not to look at the ugly code, you can find occasionally updated snapshots at the project home page, together with examples, screenshots, introductory papers and slides.

The most important outstanding item, apart from minor improvements of the GUI, is support for hierarchical HCPN models. While that would allow HCPN to be used for modelling concurrent systems or for teaching concurrency concepts, my personal interest is in exploring the design space beyond the basic hybrid formalism. This is currently a personal hobby project, so progress will depend on demand and funding.

Further reading

- Project home:
  http://www.cs.kent.ac.uk/~cr3/HCPN/
- Petri Nets home:
  http://www.daimi.au.dk/PetriNets/

3.3 Type System/Program Analysis

3.3.1 Chameleon

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

Chameleon is an experimental version of Haskell which incorporates a user-programmable type system based on Constraint Handling Rules (CHRs). Chameleon programs are compiled to plain Haskell, i.e. can be executed by any standard Haskell system such as GHC etc.

Latest developments

Improved Inference for Checking Type Annotations We consider type inference in the Hindley/Milner system extended with type annotations and constraints with a particular focus on Haskell-style type classes. We observe that standard inference algorithms are incomplete in the presence of nested type annotations. To improve the situation we introduce a novel inference scheme for checking type annotations. Our inference scheme is also incomplete in general but improves over existing implementations as found e.g. in the Glasgow Haskell Compiler (GHC). For certain cases (e.g. Haskell 98) our inference scheme is complete.

Unifying GRDTS and type classes with existential types We present a formal framework for existential types and type classes. In contrast to Laeufer's original proposal our system includes multi-parameter type classes and functional dependencies etc. Our system is powerful enough to express guarded recursive data types, a recent extension of algebraic data types. Type inference in our extension becomes a hard problem. For this purpose, we introduce a novel constraint solver. In general, we lose the principal types property. However, we consider several classes for which we infer a principal type if one exists.

Both extensions have been implemented as part of the Chameleon system.

Further reading

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

3.3.2 Constraint Based Type Inferencing at Utrecht

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

With the generation of understandable type error messages in mind we have devised a constraint based type inference method in the form of the Top library. This library is used in the Helium compiler (for learning Haskell) (→ 2.6.2) developed at Universiteit Utrecht. Our philopsophy is that no single type inferencer works best for everybody all the time. Hence, we want a type inferencer adaptable to the programmer's needs without the need for him to delve into the compiler. Our goal is to devise a library which helps compiler builders add this kind of technology to their compiler.

The main outcome of our work is the Top library which has the following characteristics:

- It uses constraints to build a constraint tree which follows the shape of the abstract syntax tree.
- These constraints can be ordered in various ways into a list of constraints
- Various solvers (specifically a fast greedy one, a slower global one, and the chunky solver which combines the two) exist to solve the resulting list of constraints.
- The library is easily extended with new constraints, and the type graph implementation includes various heuristics to find out what is the most likely source of an inconsistency. Some of these heuristics are very general, others are more tailored towards Haskell. Some the heuristics are fixed, like a majority heuristics which takes into account that there is 'more' evidence that a certain constraint is the root of an inconsistency. In addition, there are also heuristics specified from the outside. By means of a siblings directive, a programmer 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.
- It preserves type synonyms as much as possible,
- We have support for type class directives. It allows programmers to for instance specify that certain instances will never occur. The type inferencer can use this information to give better error messages. Other directives can be used to specify additional invariants on type classes. For instance, that two type classes do not share a common type (Fractional vs. Integral). A paper about this subject will find its way into PADL 2005. Although we have implemented this into Helium, the infrastructure applies as well to other systems of qualified types.
- The various phases in type inferencing have now been integrated by a slightly different, more general choice of constraints.

An older version of the underlying machinery for the type inferencer has been published in the Proceedings of the Workshop of Immediate Applications of Constraint Programming held in October 2003 in Kinsale, Ireland.

The entire library is parameterized in the sense that for a given compiler we can choose which information we want to drag around.

The library has been used extensively in the Helium compiler, so that Helium can be seen as a case study in applying Top in a real compiler. In addition to the above, Helium also

- has a logging facility for building collections of correct and incorrect Haskell programs (including time line information),
- has a run-time parameters for experimenting with various solvers and constraint orderings.
- gives precise error location information,
- supports specialized type rules, which are a means to override the order in which certain expressions are inferenced and how the type error messages are formulated (see our paper presented at ICFP '03). These type rules are especially useful for making the type error messages for domain specific extensions to Haskell correspond more closely to the domain, instead of the underlying Haskell language structures. The specialized type rules are automatically checked for soundness and completeness with respect to the original type system.

Since the report of May 2004

- Bastiaan Heeren is currently writing his PhD thesis. It promises to yield a much more complete and mature picture of what we do and how we do it, including a discussion of constraint trees and type graphs.
- A student has worked on converting the strictness analysis of Glynn, Stuckey and Sulzmann into a format which can be handled by our Top library. This work is not yet mature enough to report upon.
- The type inference directives need to be improved for usability, for which we have plenty of ideas, but these still have to be implemented and reported on.
- A grant has been obtained for continuing our work on the subject.

Further reading

Project website:
http://www.cs.uu.nl/groups/ST/Center/Top

3.3.3 EHC, 'Essential Haskell' Compiler

Report by:	Atze Dijkstra
Participants:	Atze Dijkstra, Doaitse Swierstra
Status:	active development

The purpose of the EHC project is to provide a description a Haskell compiler which is as understandable as possible so it can be used for education as well as research.

For its description an Attribute Grammer system is used as well as other formalisms allowing compact notation like parser combinators.

The EHC project also tackles other issues:

- In order to avoid overwhelming the innocent reader, the description of the compiler is organised as a series of increasingly complex steps. Each step corresponds to a Haskell subset which itself is an extension of the previous step. The first step starts with the essentials, namely typed lambda calculus.
- Each step corresponds to an actual, that is, an executable compiler. Each of these compilers is a compiler in its own right so experimenting can be done in isolation of additional complexity introduced in later steps.
- The description of the compiler uses code fragments which are retrieved from the source code of the compilers. In this way the description and source code are kept synchronized.

Currently EHC already incorporates more advanced features like higher-ranked polymorphism, partial type signatures, class system, explicit passing of implicit parameters (i.e. class instances), extensible records, kind polymorphism.

Part of these features has been described at the recent AFP summerschool (→ 1.5.1) (lecture notes yet to appear, handouts are available).

The compiler is used for small student projects as well as larger experiments such as the incorporation of an Attribute Grammar system.

Our plans for the near future are to complete the description of all steps.

We also hope to provide a Haskell frontend dealing with all Haskell syntactic sugar left out of EHC.

Further reading

- Homepage:
  http://www.cs.uu.nl/groups/ST/Ehc/WebHome
- AFP (→ 1.5.1) handouts:
  http://www.cs.uu.nl/research/techreps/UU-CS-2004-037.html
- Attribute grammar system:
  http://www.cs.uu.nl/groups/ST/twiki/bin/view/Center/AttributeGrammarSystem
- Parser combinators:
  http://www.cs.uu.nl/groups/ST/Software/UU_Parsing/

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 (→ 4.3.2) to generate a framework for generic programming on terms. A new release has been announced in May 2004.

Languages

Light-weight generic programming Generic functions for data type traversals can (almost) be written in Haskell itself, as shown by Ralf Laemmel 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, see the recently published (in ICFP '04) paper "Scrap more boilerplate: reflection, zips, and generalised casts" available from http://www.cs.vu.nl/boilerplate/. This paper shows how to fill some of the

gaps (such as generic zips) which previously were difficult to solve in this approach.

In "Generics for the masses", ICFP '04, Ralf Hinze shows how to write generic programs in Haskell98, without any fancy extensions. See http://www.informatik.uni-bonn.de/~ralf/.

Generic Haskell Andres Löh successfully defended his PhD thesis "Exploring Generic Haskell" on September 2, 2004. The thesis describes Dependency-style Generic Haskell, and introduces, amongst others, a new type system for Generic Haskell that at the same time simplifies the syntax and provides greater expressive power. Electronic copies are available at http://www.cs.uu.nl/~andres/ExploringGH.pdf.

Generic Haskell is used in "UUXML: A Type-Preserving XML Schema – Haskell Data Binding" by Frank Atanassow, Dave Clarke and Johan Jeuring, PADL '04, to implement a Haskell-XML data binding from XML Schemas to Haskell. Furthermore, Atanassow and Jeuring show how to use this data binding together with legacy code in "Inferring Type Isomorphisms Generically", in MPC '04.

Ulf Norell and Patrik Jansson at Chalmers show how to prototype generic programming in Template Haskell in a paper with the same title in MPC 2004. Ulf Norell has won the Succ Zeroth IOHCC (→ 1.5.1) with an obfuscated generic program.

The code generated by Generic Haskell, PolyP, and Clean contains many conversions between structure types and data types, which slows down the generated code. To remove these conversions, a special-purpose partial evaluator has to be written. Alimarine and Smetsers show how to do this (for Clean) in Optimizing generic functions in MPC'04.

Current Hot Topics

Generic Haskell: incorporating views on data types in the language; binding XPath to Haskell, using generic programming for validating XPath expressions against a Schema. Other: the relation between generic programming and dependently typed programming; the relation between coherence and generic programming; better partial evaluation of generic functions; methods for constructing generic programs.

Major Goals

Major Goals: Efficient generic traversal based on type-information for premature termination (see the Strafunski project (→ 4.3.2)). 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/

There is a mailing list for Generic Haskell: ⟨generic-haskell@generic-haskell.org⟩. See the homepage for how to join.

3.5 Arrow notation

Report by:	Ross Paterson
Status:	stable

Arrow notation has been supported by GHC since release 6.2, and was used by John Hughes in the Advanced Functional Programming School this year (→ 1.5.1). Both the GHC implementation and the arrow preprocessor are actively maintained.

Several people have applications that are slightly more constrained than the Arrow class, but would still like to use arrow notation. Amr Sabry, Josef Svenningsson, Peter Gammie, Simon Peyton Jones and I are discussing possible extensions. It looks complicated.

Further reading

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

4 Libraries

4.1 Packaging and Distribution

4.1.1 Hackage and Cabal (formerly the Library Infrastructure Project)

Report by: Isaac Jones

Background

Hackage (Haskell Package) is an effort to provide a framework for developers to more effectively contribute their software to the Haskell community. The Haskell Cabal (Common Architecture for Building Applications and Libraries) is one aspect of that effort.

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, tools, 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

Hackage and The Haskell Cabal seek 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

We have made an alpha release of the first phase, Cabal, the common build system, and some prototype tools for managing package collections and for building OS packages (for Debian) have been implemented on top of this. There are a number of real libraries and tools included as examples (HUnit (→ 5.4.5) and WASH (→ 4.7.4), for instance).

Further reading

- http://www.haskell.org/cabal
- http://www.haskell.org/cabal/proposal/

4.1.2 LicensedPreludeExts

Report by: Shae Erisson

The PreludeExts wiki page started with an oft-pasted email on the `#haskell` IRC channel, where at least once a week someone asked for a permutations function. That sparked a discussion of what code is missing from the Prelude, once the wiki page was started, submissions poured in, resulting in a useful and interesting collection of functions. Last year's PreludeExts has become this year's BSD LicensedPreludeExts since John Goerzen wanted to have explicit licensing for inclusion into debian packages. If you contributed code to PreludeExts and haven't yet moved it to LicensedPreludeExts, please do so!

http://www.haskell.org/hawiki/LicensedPreludeExts

4.1.3 Haskel User Submitted Libraries (haskell-libs)

Report by: Shae Erisson

haskell-libs is slowly being migrated off of sourceforge and into various darcs repositories on ScannedInAvian.org. You can see how it's going on http://www.ScannedInAvian.org/cgi-bin/darcs.cgi.

4.2 General libraries

4.2.1 Pesco.Cmdline – a command line parser ≠ GNU getopt

Report by: Sven Moritz Hallberg

This is a useful module for handling command line options of the familiar (`-hello`) form. It has some nifty features and I think it's more convenient for the quick every-day standard use than System.GetOpt.

The website above also contains some other things, but should be easy enough to navigate. Specifically, in the "Pesco.Cmdline" section, the distribution tarball is under the link named [dist] and [man] links to the hypertext reference documentation.

The module is a literate program and comes with a complete set of manpages (Yes, real Unix manpages!). See the [doc] link for a PDF rendition of the module itself, [man] for the hypertext manpages, and [ref] for the manpages in PDF (formatted for printing as an A5 booklet).

As of yet, Cmdline does not support explicitly reporting errors, it always calls `error`. I will add that shortly, however. Also, it is currently not possible to ignore unrecognized command line arguments (for chaining command line parsers) or errors in general. The next version will expose an additional lower-level interface on which such functionality can be built easily.

In the following, I quote a short comparision between System.GetOpt and Pesco.Cmdline which I wrote in reply to a request from the mailing list:

GetOpt's basic idea, given a type `alpha`, is to parse each command line option into a value of type `alpha`. Options to be recognized are specified in "none", "mandatory", and "optional" argument variants. For the latter two, a function must be given for mapping the option argument (`::  String`) to the `alpha` value.

This means, in the typical use case, you would define that type `alpha` to represent your different command line options. Then you write suitable readers for the option arguments. If you want explicit reporting of errors in the arguments, you include values in `alpha` to represent them as well.

After you call the getOpt function, you need to go through the resulting list of `alpha`s to react to your options.

In the Cmdline module, the type `alpha` is not needed. You again specify the options you want to recognize, stating whether they should take arguments or not. In contrast to GetOpt, each command line option is mapped to a "program parameter", which is, conceptually, just a named value of some type that is made accessible by calling the `get_args` function. The parameter's value is determined by the presence of a corresponding command line option. If the option is not there, a default value is assumed.

Instead of thinking about command line options I want my program to accept, I like to think about runtime parameters it should depend on. I specify those in a list to `get_args` , which determines their values. Here, boolean parameters correspond to command line *flags* (given or not given – default `False`). Options with mandatory arguments can represent parameters of pretty much any type. Those with optional arguments represent types of the form `Maybe a` – `Nothing` if not given, `Just d` if given without argument (`d` is a default value) and `Just x` if given with argument (`x` is the argument value).

To access the parameter values in the program, `get_args` returns a function mapping parameter names to their value, type `forall a. (Typeable a) => String -> a`. Each parameter can have several names which correspond directly to the names for the command line options. For instance, if I specify a boolean parameter to be recognized under the names "`v`" and "`verbose`", `get_args` will accept the flags "`-v`" and "`-verbose`" as a consequence. If the function it returns is called `parm`, I can access that parameter as `parm "v"` or `parm "verbose"`. Of course the type is checked to match. Parsing of the option arguments also happens automatically.

Further reading

http://www.scannedinavian.org/~pesco/

4.2.2 System.Time: a redesigned Time library

Report by:	Simon Marlow
Status:	stalled

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 textttClockTime
- the underspecified behaviour of `TimeDiff` / `diffClockTimes` / `addToClockTime`

The latest proposal was posted to the libraries@haskell.org mailing list in July 2003, 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.

4.2.3 A redesigned IO library

Report by:	Simon Marlow

Some time ago on the libraries mailing list there was a discussion about a replacement for Haskell's 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 deficiency in Haskell's current IO library, namely the lack of support for specifying a character translation.

- More features: e.g. mapped file support.

See the libraries archives for the discussion, e.g.

- 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

Since the previous report some progress has been made on a prototype, which is available here: http://haskell.org/~simonmar/new-io.tar.gz.

The prototype currently supports only basic I/O using files, but has some support for internationalization. I (Simon M.) am not actively working on this at the moment, so anyone that would like to pick this up is entirely welcome.

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

Report by:	Simon Marlow

The System.Process library is now complete, and will be available in the next release of GHC (6.4). In the meantime, the implementation can be found in CVS, in fptools/libraries/base/System/Process.hs.

4.2.5 The Haskell Cryptographic Library

Report by:	Dominic Steinitz

The current release is 1.2.2. New, since the last report, is the addition of AES courtesy of Lukasz Anforowicz and the inclusion of a module to support large words (Word128, Word192 and Word256) for use as keys.

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

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

The library now supports: DES, Blowfish, AES, Cipher Block Chaining (CBC) mode, PKCS5 and nulls padding, MD5, SHA-1, RSA, OAEP, ASN.1 and PKCS#8.

The library follows the hierarchical standards and has Haddock (→ 5.5.5) style documentation. There are demo / test programs using published test vectors and instructions on how to use RSA in Haskell and interwork with openssl. In particular, you can generate key pairs using your favorite method (openssl, for example) and then use them in Haskell. A big improvement on previous versions is the ability to read the private key into your Haskell program via PKCS#8 and use it to decrypt something encrypted with your public key.

There is still plenty of existing code that should be incorporated such as RC4 (courtesy of Doug Hoyte). Shawn Garbett is looking at suppporting X.509 certificates which should allow the use of RSA for encryption and also the use of signatures. With this and PKCS#12, the library should not need much more in the way of addition. Future work includes providing PKCS#12 support, re-writing the ASN.1 module and using Cabal to package the library.

Further reading

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

4.2.6 Numeric prelude

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

The hierarchy of numerical type classes is revised and oriented at algebraic structures. Axiomatics for fundamental operations are given, superfluous superclasses like Show are removed, semantic and representation-specific operations are separated, the hierarchy of type classes is more fine grained, and identifiers are adapted to mathematical terms. Both new types (like power series and values with physical units) and type classes (like the VectorSpace multi type class) are introduced. Using the revisited system requires hiding some of the standard functions provided by Prelude, which is fortunately supported by GHC.

Future plans

Collect more Haskell code related to mathematics, e.g. for linear algebra. Study of alternative numeric type class proposals and common computer algebra systems.

Further reading

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

4.2.7 Haskore revision

Report by:	Henning Thielemann
Status:	experimental, active development

Haskore is a set of Haskell modules by Paul Hudak that allow music composition within Haskell, i.e. without the need of a custom music programming language. In general this project aims at improving consistency throughout the package, revising design decisions, fixing bugs, and eventually extending Haskore. In particular some improvements are: The Music structure is based on a more general Media data structure as proposed by Paul Hudak. The core Music data structure is hidden by functions that work on it. The support for infinite Music objects is improved. It is possible to combine music composition with audio signal processing, i.e. it is possible to create audio streams of music entirely with Haskell code. There is a test suite for checking various functions of Haskore.

Future plans

Introduce a more general notion of instruments which allows for more parameters that are specific to certain instruments. Allow modulation of music similar to the controllers in the MIDI system. Connect to other Haskore related projects. Adapt to the Cabal (→ 4.1.1) system.

Further reading

- http://www.haskell.org/hawiki/Haskore
- http://cvs.haskell.org/darcs/haskore/

4.2.8 Yampa

Report by:	John Peterson and Henrik Nilsson

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 version of Yampa 0.4 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, and performs physical modeling of mobile differential-drive robots equipped with several kinds of sensors. A pre-configured version of the simulator allows one to play RoboCup Soccer.
- A tutorial (from the *2002 Summer School on Advanced Functional Programming*, Oxford, UK).
- The Space Invaders game from the 2003 Haskell workshop.

This release adds a BSD style license to the system and fixes a few minor bugs.

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.

Thanks to Abraham Egnor for contributing cairo binding, which uses Yampa for reactive animation. Download instructions are at http://www.cairographics.org/hscairo.

Further reading

http://www.haskell.org/yampa

4.2.9 The revamped monad transformer library

Report by:	Iavor Diatchki
Status:	mostly stable

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 similar 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.

The monad transformer library now has its first official release. I have put it on my web page: http://www.cse.ogi.edu/~diatchki/monadLib

It is in many ways similar to what's distributed with GHC/Hugs/etc, but I think also simplified and better organized. The library intraface is documented

with haddock (→ 5.5.5). The monads/transformers currently in the library are:

- ReaderT (environment)
- WriterT (output)
- StateT
- ExceptT
- BackT
- ContT

In this version I decided to implement some of the transformers (backtracking,exceptions) in continuation passing style, thinking that they may work better that way. I haven't done any formal testing on that though. The Haskell extensions the library uses are:

- Multiparameter classes (important).
- Rank-2 polymorphism (for the CPS implementations, could be removed).
- Functional dependencies (could be removed, but is likely to requre more type annotations).

For any questions, comments, or bug reports please send me a mail.

Further reading

http://www.cse.ogi.edu/~diatchki/monadLib

4.2.10 HBase

Report by:	Ashley Yakeley

HBase is a large collection of library code, compiled "`-fno-implicit-prelude`", intended as an experimental/alternative reorganized interface to the existing standard libraries making full use of GHC's extensions. HBase development is driven by HScheme (→ 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".

Very little work is currently being done on it, as the main developer's free time has been shortened by gainful employment. Further work may resume, at a reduced pace, once left-over issues in the latest JVM-Bridge (→ 5.1.3) have been cleared up.

Further reading

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

4.2.11 Pointless Haskell

Report by:	Jorge Sousa Pinto

Pointless Haskell is a library for point-free programming with recursion patterns defined as hylomorphisms. It is part of the UMinho Haskell libraries that are being developed at the University of Minho (→ 7.1.8). The core of the library is described in "Point-free Programming with Hylomorphisms" by Alcino Cunha.

Pointless Haskell also allows the visualization of the intermediate data structure of the hylomorphisms with GHood. This feature together with the DrHylo (→ 5.2.7) tool allows us to easily visualize recursion trees of Haskell functions, as described in "Automatic Visualization of Recursion Trees: a Case Study on Generic Programming" (Alcino Cunha, In volume 86.3 of ENTCS: Selected papers of the 12th International Workshop on Functional and (Constraint) Logic Programming. 2003).

Further reading

The Pointless Haskell library is available from http://wiki.di.uminho.pt/bin/view/Alcino/PointlessHaskell.

4.2.12 hs-plugins

Report by:	Donald Bruce Stewart
Status:	active development

hs-plugins is a library for dynamic loading and runtime compilation of Haskell modules at runtime, into Haskell and foreign applications. It can be used to implement standard application plugins, hot swapping of modules in running applications, and enables the use of Haskell as an application extension language. The library has stabilised in the last six months, and we hope to release v1.0 around January 2005.

Further reading

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

4.2.13 MissingH

Report by:	John Goerzen
Status:	active development

MissingH is a library designed to provide the little "missing" features that people often need and end up implementing on their own. Its focus is on list, string, and IO features, but extends into other areas as well. The library is 100% pure Haskell code and has no dependencies on anything other than the standard libraries distributed with current versions of GHC and Hugs.

In addition to the smaller utility functions, recent versions of MissingH have added a complete FTP client parser library, a MIME type guesser, and a modular logging infrastructure, complete with support for Syslog.

Future plans for MissingH include adding more network client and server libraries, support for a generalized URL downloading scheme that will work across all these client libraries, and enhancing the logging system.

This library is licensed under the GNU GPL.

Further reading

MissingH is available from http://quux.org/devel/missingh.

4.3 Parsing and transforming

4.3.1 Parsec

Report by:	Daan Leijen
Status:	stable

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

4.3.2 Strafunski

Report by:	Joost Visser
Status:	active, maintained, new release in October 2004
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 (→ 3.4);
- the library ATermLib for data exchange;
- the tool Sdf2Haskell (→ 5.2.5) for external parser and pretty-print integration.

The Strafunski-style of generic programming can be seen as a lightweight variant of generic programming (→ 3.4) because no language extension is involved, but generic functionality simply relies on a few overloaded combinators that are derived per datatype. By default, Strafunski relies on DrIFT to derive the appropriate class instances, but a simple switch is offered to rely on the "Scrap your boilerplate" (→ 3.4) model as available in the Data.Generics library.

The Sdf2Haskell component of Strafunski has recently been extended to offer not only parsing support via the external "sglr" parser, but also:

- parsing support via HaGLR (→ 5.2.6), an experimental 100% Haskell implementation of Generalized LR parsing
- pretty-printing support, based on the pretty-print combinators as available in the Text.PrettyPrint.HughesPJ library. The generated pretty-printers are functional strategies that offer uniform behaviour which can be customized with type-specific behaviour.

Strafunski is used in the HaRe project (→ 5.3.4) and in the UMinho Haskell Libraries (→ 7.1.8) to provide analysis and transformation functionality for languages such as Java, VDM, SQL, spreadsheets, and Haskell itself.

Further reading

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

4.3.3 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 visualization 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 visualization mechanisms to allow exploring such temporal data. These visualization 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.

Following on from the case studies we have improved and extended the visualization systems and implemented some of the ideas from the case studies. Demos and screenshots are available on the Medina webpage: http://www.cs.kent.ac.uk/~cr24/medina.

Currently there is no released version of the Medina library, but my PhD thesis has been submitted so I am now in the process of preparing a release. This should be available real-soon-now.

4.4 Data handling

4.4.1 DData

Report by:	Daan Leijen
Status:	stable

DData is a library of efficient data structures and algorithms for Haskell (Set, Bag, and Map). It is actively maintained and stable.

DData is currently under review for inclusion in the standard hierarchical module name space, and you are invited to join the discussion on the Haskell libraries mailing list.

The current proposal is maintained by J.P. Bernardy and can be found at: http://users.skynet.be/jyp/DData/doc and http://users.skynet.be/jyp/DData/ddata.tar.gz

Further reading

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

4.4.2 A library for strongly typed heterogeneous collections

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

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

An immediate application of HLists is the implementation of open, extensible records with first-class, reusable labels. We have also used HList for type-safe database access in Haskell. The HList library relies on common extensions of Haskell 98.

We are currently working on applications of HList to a powerful object-oriented system and to expressive type-level programming.

Further reading

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

4.4.3 HSQL

Report by:	Krasimir Angelov
Status:	stable

The HSQL is a simple library for database access from Haskell. It is relatively small and complete. bug fixes are always welcome and If someone is wishing to add a new backend I will be glad to help him.

Further reading

http://htoolkit.sourceforge.net/

4.4.4 Takusen

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

Takusen is a library for accessing DBMS's. It is a low-level library like HSQL (→ 4.4.3), in the sense that it is used to issue SQL statements. Takusen's 'unique-selling-point' is a design for processing query results using a left-fold enumerator. For queries the user creates an iteratee function, which is fed rows one-at-a-time from the result-set. We also support processing query results using a cursor interface, if you require finer-grained control.

Takusen is under active development, although progress is slow. Since the last HC&A Report we have added support for Sqlite, re-organised the library modules, addressed some performance problems, and improved the Oracle connection code (OS-authenticated logon is possible now). We plan to continue adding implementations for other DBMS's, and support for bind variables. We're also reviewing the current design, with a view to doing everything in the IO monad, rather than using monad transformers. We expect this to make the code (both library and user) a bit simpler; users won't have to use liftIO to perform IO actions inside database actions.

http://cvs.sf.net/viewcvs.py/haskell-libs/libs/takusen/

4.4.5 HaskellDB

Report by:	Anders Höckersten
Status:	active development and maintenance
Portability:	GHC, Hugs, multiple platforms

HaskellDB is a library for accessing databases through Haskell in a type safe and declarative way. It completely hides the underlying implementation and can interface with several popular database engines through either HSQL (→ 4.4.3) or wxHaskell (→ 4.5.2). HaskellDB was originally developed by Daan Leijen. Development was restarted as part of a student project at Chalmers University of Technology. This project is now over, but several of the original project members are still actively developing and maintaining HaskellDB. We do welcome new developers and patches, as all of us are full-time students.

The current version supports:

- Completely type safe queries on databases
- Support for MySQL, PostgreSQL, SQLite and ODBC through HSQL
- Support for ODBC through wxHaskell
- Automatic conversion between Haskell types and SQL types
- Support for bounded strings
- Dynamic loading of drivers via hs-plugins (→ 4.2.12)

Future possible developments include:

- Support for more backends (Oracle)
- Support for non-SQL backends
- Driver-specific code generation. This is needed for non-SQL backends, and we have discovered that no SQL databases implement the standard in quite the same way

Further reading

http://haskelldb.sourceforge.net

4.5 User interfaces

4.5.1 The Common GUI API effort

Report by:	Axel Simon

The usefulness of the Haskell language depends crucially on the provided libraries. In particular, efforts to write an application with a graphical user interface has long been complicated by the large number of mostly incomplete libraries (or research prototypes). In spring 2003 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). At the Haskell Workshop 2003 a quick poll revealed that 1/3 of the people thought that this major undertaking is worthwhile, 2/3 thought that a new binding to a readily available cross-platform approach is adequate. Hence the CGA idea was not pursued and wxHaskell, a binding to wxWidgets, is recommended for new developments. Other libraries might of course continue to exist, in particular if they fill a niche for some applications.

4.5.2 wxHaskell

Report by:	Daan Leijen
Status:	beta, 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 wxWidgets – 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 wxWidgets 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 90% of the wxWindows functionality – about 3000 methods in 500 classes with 1300 constant definitions. wxHaskell has been built with GHC 6.x on Windows, MacOS X and Unix systems with GTK, and binary distributions are available for common platforms.

Since the last community report, most work has been directed into improved stability and a better build system. There is also better integration with other packages: HaskellDB (→ 4.4.5) works with the wxHaskell ODBC binding and HOpenGL (→ 4.6.2) can work with the OpenGL canvas. The wxHaskell website also shows some screenshots of larger sized applications that are developed with wxHaskell. It is most satisfying to see that even those larger applications are ported without any real difficulties – Haskell is becoming a very portable language indeed!

Current work is directed at improving documentation and stability across platforms, and we hope to release the 1.0 version in January 2005. In between there will be a 0.9 release candidate with full printing 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. See also "wxHaskell: a portable and concise GUI library", Daan Leijen, Haskell workshop 2004.

4.5.3 FunctionalForms

Report by:	Sander Evers

FunctionalForms is a combinator library built on top of wxHaskell (→ 4.5.2) which enables a concise and declarative programming style for *forms*: dialogs which only show and edit a set of values (used in many applications as *Options* or *Properties* dialogs). Control and layout definition are combined into one expression, there's no IO monad programming, and values are passed to and from the controls almost automatically. Still, the type of the edited values and the layout structure can be managed independently, thanks to a new programming technique called *compositional functional references*. Currently, FunctionalForms is still in an early state; I plan to extend it. Feedback is welcomed.

Further reading

http://www.cs.ru.nl/~sandr/FunctionalForms

4.5.4 HToolkit

Report by:	Krasimir Angelov

The HToolkit is a platform independent package for Graphical User Interfaces. The package is split into two libraries GIO and Port. The Port is a low-level Haskell 98+FFI (→ 3.1) compatible API, while GIO is a highlevel 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.

There are some new things. There is a better support for menus and toolbars under both Windows and Linux. There is also new API which allows to create action based menu items and toolbar buttons. The "action" here is something like GtkAction widget but it is at Haskell level and it is available for both Windows and Linux. There isn't a new release yet.

Further reading

http://htoolkit.sourceforge.net/

4.5.5 gtk2hs – A binding to the Gtk GUI library version 2.0–2.4.

Report by:	Duncan Coutts
Maintainer:	Axel Simon

This project provides a high-quality binding to the Gtk+ GUI library together with some Gnome extensions (at the moment Glade, GConf and a source code editor widget).

Compared to wxHaskell (→ 4.5.2) it does not adopt its look and feel to different platforms, however, it allows greater integration with its native Gtk+/Gnome environment. It also has automatic memory management (wxWidgets does not provide proper support for garbage-collected languages).

The library is actively maintained and developed. In particular we are planning to add more Gnome widgets, support for embedding the Mozilla rendering engine and add a medium level API layer similar to that of wxHaskell. Since the last HC&A Report, we have achieved almost complete coverage of the Gtk+ API, improved Haddock (→ 5.5.5) documentation and have begun conversion to using hierarchical module names.

The latest release of gtk2hs, version 0.9.6, is known to run on Linux, FreeBSD, MacOS X, Windows and Solaris. Packages are currently available for Fedora Core (→ 7.2.2) and Gentoo (→ 7.2.4).

4.5.6 HTk

Report by:	Christoph Lüth and George Russell
Status:	stable, 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 concurrent manner. HTk is known to run under Linux, Solaris, FreeBSD, Windows (98, 2k, XP) and will probably run under many other POSIX systems as well. It works with GHC, version 6.0 and up.

Further reading

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

4.5.7 Fudgets

Report by:	Thomas Hallgren

Fudgets is a GUI toolkit designed and implemented by Magnus Carlsson and Thomas. Most of the work was done in 1991–1995, and the library has been in minimal maintenance mode since then. It compiles with recent

versions of GHC (e.g., GHC 6.2.1) on many Unix-like platforms (Linux, SunOS, Mac OS X, etc).

For documentation and downloads, see: http://www.cs.chalmers.se/Fudgets/.

Recent snapshots can also be found at: http://www.cse.ogi.edu/~hallgren/untested/.

Two applications using the Fudgets:

- The proof assistant Alfa, http://www.cs.chalmers.se/~hallgren/Alfa/
- The Programatica Haskell Browser, http://www.cse.ogi.edu/~hallgren/Programatica/

4.6 Graphics

4.6.1 HSX11, HGL, and Win32

Report by:	Ross Paterson

The hierarchical libraries include a new version of the X11 package that no longer requires GreenCard (→ 5.1.1). The Haskell Graphics Library works on X11, and also includes a compatibility module for use with Paul Hudak's "The Haskell School of Expression". Both packages will be included in the next releases of GHC and Hugs. There is also a Win32 package that no longer requires GreenCard, but it is in urgent need of a volunteer to get it working and maintain it. If that were done, HGL could be expected to work on both platforms.

4.6.2 HOpenGL – A Haskell Binding for OpenGL and GLUT

Report by:	Sven Panne
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 or wxHaskell (→ 4.5.2) on the other hand is a goal, see e.g. http://wxhaskell.sourceforge.net/samples.html#opengl

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 Haskell 98 + FFI (→ 3.1)
- No GreenCard (→ 5.1.1) dependency anymore
- Full OpenGL 1.5 support (NURBS currently only partly implemented)
- A few dozen 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 (→ 7.2.3), and Mac OS X.

The binding comes with a lot of examples from the Red Book and other sources, and Sven Eric Panitz has written a tutorial using the new API (http://www.tfh-berlin.de/~panitz/hopengl/), so getting started should be rather easy.

Further reading

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

4.6.3 Pancito

Report by:	Andrew Cooke
Status:	not currently being developed further

Version (2.2) of Pancito (a functional images toolkit initially inspired by Pan) is available at http://www.acooke.org/jara/pancito – it extends 2.1 with useful output options (a progress meter and the possibility to preview a small area of an image) and better structured code (points are now a typeclass, allowing now kinds of coordinates to be added more easily, for example).

Art generated with Pancito can be seen in the gallery (generated with Halipeto (→ 4.7.1)) at http://www.acooke.org/pancito.

4.7 Web and XML programming

4.7.1 Halipeto

Report by:	Andrew Cooke
Status:	not currently being developed further

Halipeto generates web pages from templates (much like JSP, Zope TAL etc). It is written in Haskell (with a ghc extension) and is available from http://www.acooke.org/jara/halipeto.

Since Haskell functions are directly associated with element attributes, the system is flexible and easy to

extend (providing you can program in Haskell). An example site generated using Halipeto, containing some Pancito (→ 4.6.3) images, is at http://www.acooke.org/pancito; the code for that site is included in the Halipeto download as an example.

Content management systems typically have four distinguishing features: server integration with dynamic page generation; a template engine for generating output in the correct format; a database for storing content; and a dynamic web interface for managing the system. Halipeto only supplies two of these – templates and a simple file-system based database. Without further work it can only be used, therefore, to generate static web pages from the database (SQL integration should be fairly simple as the IO Monad is already threaded through the system).

Thanks to HaXml (→ 4.7.2) for the XML support (Halipeto includes the relevant files, HaXml does not need to be installed).

4.7.2 HaXml

Report by:	Malcolm Wallace
Status:	stable, maintained

HaXml provides many facilities for using XML from Haskell. The public release is currently at version 1.12. Graham Klyne (→ 8.2) has recently done a lot of work to extend HaXml for namespaces, better Unicode support, and much more. His version is currently separate, but we hope eventually to merge those contributions back into the main HaXml tree.

Further reading

- http://haskell.org/HaXml
- http://www.ninebynine.org/Software/HaskellUtils/

4.7.3 Haskell XML Toolbox

Report by:	Uwe Schmidt
Status:	fourth major release (current release: 4.02)

Description

The Haskell XML Toolbox is a collection of tools for processing XML with Haskell. It is itself purely written in Haskell 98.The core component of the Haskell XML Toolbox is a validating XML-Parser that supports almost fully the Extensible Markup Language (XML) 1.0 (Second Edition),

The Haskell XML Toolbox bases on the ideas of HaXml (→ 4.7.2) 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
- hierarchical library support
- package support for ghc
- native Haskell support of HTTP 1.1 and FILE protocol
- HTTP and access via other protocols via external program curl
- tested with W3C XML validation suite
- example programs

Current Work

- a complex example application using the HXT library and the wxWidgets framework has been developed. This program consists of an editor for a web photo album. The album itself is represented as XML document.

- a simple and more flexible interface to the HXT DOM and the existing filter based on the Arrow interface are under construction.

- a project for supporting the Relax NG XML schema definition for validation ă has started this autumn and will be finished in September 2005.

- in September 2004 a master student has started his thesis work. In this project a “HXT cookbook” will be developed. users guide. In this user guide the development of a nontrivial example application will be described, for demonstrating the programming technics with filters and their combinations on real life problems.

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 documentation is a bit out of date. This is one reason for the users guide project.

4.7.4 WASH/CGI – Web Authoring System for Haskell

Report by: Peter Thiemann

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

- complete interactive 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

Completed Items are:

- much improved and documented preprocessor for translating markup in XML syntax into WASH/HTML
- package-ifycation of WASH (& much simpler installation)
- caching of documents (but turned off by default)

Current work includes

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

Further reading

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

4.7.5 GXS – The Generic XML Serializer

Report by: Simon Foster
Status: beta

GXS allows purpose-built record syntax Haskell data-types to serialized to and deserialized from XML in a highly extensible manner. For example

```
data Person =
  Person{surname::String, forename::String,
         a_nums::[Int], a_isMale::Bool}
    deriving (Typeable, Data, Show)
```

is a simple data-type with elements surname and forenames and two attributes, note that fields default to elements if no letter prefix is supplied. After attempting to implement a serializer using type-classes, and finding it to be impossible (see Haskell mailing-list posts in August "XML Serialization and type constraints" http://www.mail-archive.com/haskell@haskell.org/msg15180.html), after much needed advice from Ralf Lämmel (the author of the two Scrap Your Boilerplate (→ 3.4) papers), I switched to a more Generic Solution using Data.Generics. The new serializer allows record-syntax data-type serialization to be extended by the use of encoder/decoders which allows specific data-types to be encoded in different ways to the normal serialization method. For example a date (data-type with three Int fields; day, month and year) would normally (under standard GXS rules) be serialized as

```
<day>11</day><month>10</month><year>2004</year>
```

but with the help of an encoder this can instead be stored as simply `11-10-2004`, this of course is only a simple example and there are many ways in which encoders can be used, including serialization of SOAP Arrays.

The serializer also allows the insertion of hooks into the serialization process. These are different to encoders in that rather than working on the data itself, they allow meta data to be inserted into the tree, such as namespaces and XSD (or other) type data. The addition of specialized encoders and hooks is made easier by use of a simple Dynamic database which is used to store various data about the tree, such as name-space tables and type lookups, but can theoretically be tooled to any application. GXS is currently in a highly beta (not to mention messy!) state in the HAIFA CVS (https://savannah.nongnu.org/projects/haifa/) repository, but has been already used extensively in the further parts of HAIFA (it was originally written as a means to an end, rather than a project in itself, but has somewhat developed from there). The serializer can be found in haifa/src/Text/XML/Serializer.hs.

4.7.6 XML Schema

Report by: Simon Foster

The mapping of XML Schema data-types to Haskell is of paramount importance to the development of Web-Service based applications in Haskell. The original HAIFA (see last HCAR) somewhat avoided this issue by having a centralized type-mapping database, which allowed different types from XSD to be built and used manually. However this solution could never be permanent, and so it has been necessary to move on to set an XML Schema seed in Haskell. The first task (and possibly most time-consuming) was to create a set of XML Schema data-types in Haskell. Using the GXS (→

4.7.5) data-type encodings the data-types were created along with a basic (and very, very much incomplete) type-mapper, mainly for use in WSDL. The XML Schema library now utilizes Haskell's module name-spaces to create proper XML style name-spaces (http://www.w3.org/TR/REC-xml-names/), similar to those in Java; e.g. Org.W3.N2001.XMLSchema, which makes accessing types much easier for further applications. Now that HXT (→ 4.7.3) is also under the hierarchical library tree, development of derivative projects will be far more organized and usable. This work is also available in the Haskell CVS; specifically under haifa/src/Text/XML/Schema/ and particularly haifa/src/Text/XML/Schema/Org/W3/N2001/XMLSchema.hs.

4.7.7 SOAP/1.1 and WSDL/1.1

Report by:	Simon Foster

The ultimate aim of putting together GXS (→ 4.7.5) and XML Schema (→ 4.7.6) was to create a (proper) implementation of Web-Services in Haskell. GXS simply means that SOAP is just a couple of data-types with a few encodings + hooks, and of course HTTP is driven by HTTP library improved by Bjorn Bringert (→ 4.7.8). This only leaves WSDL, which is actually quite a task to do and I won't go into all the details here. Needless to say, a WSDL document maps to a number of data-types and functions which can be used to execute the operations on the remote Service. At time of writing, the implementation of WSDL is almost complete, with a working example of a simple Web-Service off http://soapclient.com, although due to (very very) partial support for XML Schema type-mapping, in practise only a sub-set of Web-Services will work even after WSDL is complete and so further work on XML-Schema will be required.

Please Note that the HAIFA CVS is currently very much in a state of flux, a lot of stuff in there belongs to old HAIFA (as well as some newer, although still redundant code), and HAC probably doesn't work at all anymore (it will soon though because it is needed for the next project).

Further reading

- http://www.w3.org/TR/2000/NOTE-SOAP-20000508/
- http://www.w3.org/TR/wsdl

4.7.8 Haskell XML-RPC

Report by:	Björn Bringert
Status:	maintained

Haskell XML-RPC is a library for writing XML-RPC client and server applications in Haskell. XML-RPC is a standard for XML encoded remote procedure calls over HTTP. The library is actively maintained and relatively stable.

Further reading

http://www.bringert.net/haskell-xml-rpc/

5 Tools

5.1 Foreign Function Interfacing

5.1.1 GreenCard

Report by:	Alastair Reid
Status:	maintained, very stable
Current release:	3.01 (6 June 2003)
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/

5.1.2 C–>Haskell

Report by:	Manuel Chakravarty
Status:	active

C–>Haskell is an interface generator that simplifies the development of Haskell bindings to C libraries. Development in the past half year has concentrated on stabilising the current feature set; so, the current version 0.13 "Pressing Forward" of C–>Haskell contains a lot of bug fixes and has better performance than the previous version. Source and binary packages as well as a reference manual are available from http://www.cse.unsw.edu.au/~chak/haskell/c2hs/.

5.1.3 JVM Bridge

Report by:	Ashley Yakeley

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

JVM-Bridge is at version 0.3: it works on Windows and also allows the use of third-party Java libraries. A 0.3.1 release to fix Mac OS X build issues may be forthcoming.

Further reading

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

5.1.4 PHI – Python Haskell Interface

The Python-Haskell-Interface (PHI) is a pre-alpha binding between Haskell and Python. The goal is to allow writing mixed-language systems, with an eye to using Haskell components with Python systems like Zope, and taking advantage of existing Python libraries.

The binding currently supports (modulo segfaults) exposing Haskell functions as a Python module and calling Python code from Haskell. Haskell can be the main program, or linked as a Python extension module.

The code currently covers manipulating and creating Python objects, and wrapping Dynamics to be passed into Python. Marshalling classes cover some primitive types, tuples, and association lists.

I plan to add marshalling of exceptions across interlanguage calls and fix some segfaults before making an official release. My darcs repository is accessible at http://page-208.caltech.edu/phi/.

5.1.5 HOC: A Haskell to Objective-C binding

HOC is a Haskell to Objective-C binding. In a nutshell, it enables you to use Objective-C objects and frameworks from Haskell, and also enables you to write Objective-C objects in Haskell. You can write full-blown applications in HOC and use all of the Foundation, AppKit and Cocoa frameworks' classes (including all of the AppKit's GUI objects), combining Objective-C tools such as Mac OS X's Interface Builder to build the GUI graphically while writing controllers for the GUI in Haskell. You can even mix and match custom objects written in Objective-C with objects written in Haskell, depending on which language you find more suitable for the task. HOC comes some sample GUI programs: you can find screenshots of these HOC programs at http://hoc.sourceforge.net/screenshots/.

The Haskell interfaces produced by HOC are strongly typed (Objective-C classes are mapped to Haskell's typed system), can be automatically generated from Objective-C header files, and are designed to integrate well with existing Haskell code, taking advantage of features such as type classes and partial evaluation to make its Haskell API as easy to use and as 'Haskell-like' as possible. HOC's primary platform is Mac OS X, although it has been lightly tested with the free GNUstep platform on Linux. HOC requires the Glasgow Haskell Compiler (GHC) 6.2 or later.

Note: If you have heard of a Haskell to Objective-C binding named Mocha, HOC is effectively a working

version of Mocha, which was never completed due to time constraints. A previous version of HOC (0.1) was mentioned briefly on the *glasgow-haskell-users* mailing list on January 2003, but is a very different beast to the current incarnation: HOC 0.1 was more of an experiment with Template Haskell than a serious implementation. Wolfgang Thaller, the primary author of HOC, has collaborated greatly with André Pang, who was the primary author of Mocha, to forge a new HOC that we hope you will find achieves all the ambitious goals that Mocha strived for, and more. Mocha is dead, long live HOC!

HOC's webpages and source code distribution are currently available. More information on HOC (including where you can download it!) is available at: http://hoc.sourceforge.net/

5.2 Scanning, Parsing, Analysis

5.2.1 Alex version 2

Report by:	Simon Marlow
Status:	stable, maintained

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

Status: No change since the last report. The latest version is 2.0, released on August 13, 2003. Alex is in maintenance mode at the moment, and a few minor bugs reported since 2.0 have been fixed in CVS. A minor release will probably be made at some point.

Further reading

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

5.2.2 Happy

Report by:	Paul Callaghan and Simon Marlow
Status:	stable, maintained

I have recently extended Happy to produce Generalized LR parsers (based on an algorithm by Tomita). This means it can parse ambiguous grammars and produce a directed acyclic graph representing all possible parses. It is based on undergraduate project work by Ben Medlock, but has been significantly extended and improved since then. You can also attach semantic information to rules in two modes:

- to give detailed, application-specific labelling for the nodes in the DAG;
- to compute lists of overall semantic results, one per valid parse.

The latter mode can also perform monadic computations.

We plan to release a new version of Happy including the GLR extensions soon. It can be accessed now via CVS. Paul has a page http://www.dur.ac.uk/p.c.callaghan/happy-glr/ which collects useful information about this extension, plus some binaries. We have used the GLR facility in several applications, including analysis of DNA sequences and determination of correct rhythmic structures for poetry. Other possible applications include natural language and pattern analysis. The next experiment will be the GHC Haskell grammar!

Further reading

Happy's web page is at http://www.haskell.org/happy/.

5.2.3 HaLex

Report by:	Jorge Sousa Pinto

HaLeX is a Haskell library to model, manipulate and animate regular languages. This library introduces a number of Haskell datatypes and the respective functions that manipulate them, providing a clear, efficient and concise way to define, to understand and to manipulate regular languages in Haskell. For example, it allows the graphical representation of finite automata and its animation, and the definition of reactive finite automata. This library is described in the paper presented at FDPE'02.

5.2.4 LRC

Report by:	Jorge Sousa Pinto

Lrc is a system for generating efficient incremental attribute evaluators. Lrc can be used to generate language based editors and other advanced interactive environments. Lrc can generate purely functional evaluators, for instance in Haskell. The functional evaluators can be deforested, sliced, strict, lazy. Additionally, for easy reading, a colored LaTeX rendering of the generated functional attribute evaluator can be generated.

5.2.5 Sdf2Haskell

Report by:	Jorge Sousa Pinto
Maintainer:	Joost Visser

Sdf2Haskell is a generator that takes an SDF grammar as input and produces support for GLR parsing and customizable pretty-printing. The SDF grammar specifies concrete syntax in a purely declarative fashion. From this grammar, Sdf2Haskell generates a set

of Haskell datatypes that define the corresponding abstract syntax. The Scannerless Generalized LR parser (SGLR) and associated tools can be used to produce abstract syntax trees which can be marshalled into corresponding Haskell values.

Recently, the functionality of Sdf2Haskell has been extended with generation of pretty-print support. From the SDF grammar, a set of Haskell functions is generated that defines an pretty-printer that turns abstract syntax trees back into concrete expressions. The pretty-printer is updateable in the sense that its behavior can be modified per-type by supplying appropriate functions.

Sdf2Haskell is distributed as part of the Strafunski bundle for generic programming and language processing (→ 4.3.2).

5.2.6 HaGLR

Report by: Jorge Sousa Pinto and Joost Visser

HaGLR is an implementation of Generalized LR parsing in Haskell. Apart from parsing with the GLR algorithm, it supports parsing with the LR algorithm, visualization of deterministic and non-deterministic finite automata, and export of ASTs in XML or ATerm format. As input, HaGLR accepts either plain context-free grammars, or SDF syntax definitions. The SDF front-end is implemented as an extension of the Sdf2Haskell generator (→ 5.2.5). HaGLR's functionality can also be accessed as library functions, available under the Language.ContextFree subdivision of the UMinho Haskell Libraries (→ 7.1.8). HaGLR was implemented by João Fernandes and João Saraiva.

Further reading

HaGLR is available from http://wiki.di.uminho.pt/twiki/bin/view/PURe/HaGLR.

5.2.7 DrHylo

Report by: Jorge Sousa Pinto

DrHylo is a tool for deriving hylomorphisms from Haskell program code. Currently, DrHylo accepts a somewhat restricted Haskell syntax. It is based on the algorithm first presented in the paper Deriving Structural Hylomorphisms From Recursive Definitions at ICFP'96 by Hu, Iwasaki, and Takeichi. To run the programs produced by DrHylo, you need the Pointless library.

Further reading

DrHylo is available from http://wiki.di.uminho.pt/bin/view/Alcino/DrHylo.

5.3 Transformations

5.3.1 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 (→ 5.3.4).

Further reading

- The Programatica Project, overview & papers: http://www.cse.ogi.edu/PacSoft/projects/programatica/
- An Overview of the Programatica Toolset: http://www.cse.ogi.edu/~hallgren/Programatica/HCSS04/
- 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/

5.3.2 Term Rewriting Tools written in Haskell

Report by: Salvador Lucas

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

The tools have been developed at the Technical University of Valencia (UPV) as part of a number of research projects. The following people is (or has been) involved in the development of these tools: María Alpuente, Santiago Escobar, Salvador Lucas, Pascal Sotin (Université du Rennes).

Status

The previous work lead to the following tools:

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

  http://www.dsic.upv.es/~slucas/csr/termination/muterm

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

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

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

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

All these tools have been written in Haskell and use popular Haskell libraries like Parsec (→ 4.3.1) or wxHaskell (→ 4.5.2).

Immediate plans

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

References

- Abstract Diagnosis of Functional Programs M. Alpuente, M. Comini, S. Escobar, M. Falaschi, and S. Lucas Selected papers of the International Workshop on Logic Based Program Development and Transformation, LOPSTR'02, LNCS 2664:1-16, Springer-Verlag, Berlin, 2003.
- OnDemandOBJ: A Laboratory for Strategy Annotations M. Alpuente, S. Escobar, and S. Lucas 4th International Workshop on Rule-based Programming, RULE'03, Electronic Notes in Theoretical Computer Science, volume 86.2, Elsevier, 2003.
- Connecting remote termination tools M. Alpuente and S. Lucas 7th International Workshop on Termination, WST'04, pages 6-9, Technical Report AIB-2004-07, RWTH Aachen, 2004.
- MU-TERM: A Tool for Proving Termination of Context-Sensitive Rewriting S. Lucas 15th International Conference on Rewriting Techniques and Applications, RTA'04, LNCS 3091:200-209, Springer-Verlag, Berlin, 2004.

5.3.3 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/

5.3.4 Hare – The Haskell Refactorer

Report by:	Huiqing Li, Claus Reinke and Simon Thompson

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

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

A snapshot of HaRe is available from our web page, as are recent presentations from the group (including Advanced Functional Programming '04 (→ 1.5.1)), and an overview of recent work from staff, students and interns.

We expect to release a new version of HaRe including an API for users to define their own program transformations, plus documentation, by mid-November. Beyond that our aims are to support more data-oriented refactorings, to allow refactorings for systems involving the Haskell libraries and, in the longer term, to make it easier to use HaRe with GHC and its libaries. We would very much welcome feedback on the system and the API.

Further reading

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

5.3.5 VooDooM

Report by:	Jorge Sousa Pinto

VooDooM reads VDM-SL specifications and applies transformation rules to the datatypes that are defined in them to obtain a relational representation for these datatypes. The relational representation can be exported as VDM-SL datatypes (inserted back into the original specification) and/or SQL table definitions (can be fed to a relational DBMS). The first VooDooM prototype was developed in a student project by Tiago Alves and Paulo Silva. Currently, the development of VooDooM is continued in the context of the IKF-P project (Information Knowledge Fusion, http://ikf.sidereus.pt/) and will include the generation of XML and Haskell

Further reading

VooDooM is available from http://wiki.di.uminho.pt/bin/view/PURe/VooDooM.

5.3.6 LVM-OPT

Report by:	Eelco Visser and Jory van Zessen

Optimization of functional programs through strategic program transformation is still one of the projects we are pursuing in the Stratego/XT group, even if it is at a slow pace. After a year of silence, Jory van Zessen has taken over the stick from Alan van Dam and the HsOpt (see November 2003 edition of the HC&A Report) project has become the lvm-opt project; the target of optimization is LVM code produced by the Helium compiler. The goal remains to create a (full-blown) simplifier/optimizer for a lazy functional language based on rewrite rules controlled by strategies.

5.4 Testing and Debugging

5.4.1 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.

Hood and its variant GHood, for graphical display and animation, enable the user to observe the values of selected expressions in a program. Hood and GHood are easy to use, because they are based on a small portable library. A variant of Hood is built in to Hugs. Hood and GHood have remained unchanged for over two years.

HsDebug is a gdb-like debugger, that is, it is used similar to traditional debuggers for imperative languages. HsDebug is based on optimistic evaluation and hence is only available in a separate branch of GHC in CVS.

The Concurrent Haskell Debugger CHD was recently extended to support an automatic search for deadlocks.

Further reading

CHD: http://www.informatik.uni-kiel.de/~fhu/chd/

5.4.2 Hat

Report by:	Olaf Chitil and Malcolm Wallace
Status:	several recent additions; stable release forthcoming

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. The primary viewers at present allow: observation of top-level functions (hat-observe); and backwards exploration of a computation, starting from (part of) a faulty output or an error message (hat-trail). In CVS there are more viewing tools. Hat-detect is an algorithmic debugger very similar to Buddha (→ 5.4.3). Hat-explore is a new viewing tool that provides a user interface similar to traditional source-based debuggers, but allows free navigation through the trace and incorporates algorithmic debugging and program slicing. Hat-cover highlights the parts of the source program that have been executed during the computation. Hat-anim is a forward-animator showing the reduction sequence of expressions. We also have prototypes of two tools for extracting diagnostic paths from non-terminating computations. If the computation dives into a black

hole, black-hat can be used; for other forms of non-productive non-termination hat-nonterm can be used. 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, and can also be used for Haskell 98 programs that use some language extensions (FFI (→ 3.1), MPTC, fundeps, hierarchical libs).

The most recent public release of Hat (2.02) is now more than a year old and since then numerous bugfixes have been applied and several features and viewing tools added in CVS. A new release is imminent.

Further reading

http://www.haskell.org/hat

5.4.3 buddha

Report by:	Bernie Pope
Status:	active

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 and constants.

Since the last report buddha has moved to version 1.2. Currently buddha is known to work with GHC version 6 through 6.2.x. There are no plans to port it to other Haskell implementations, though there are no significant reasons why this could not be done.

Buddha is freely available as source and is licensed under the GPL. There is also a Debian package, as well as ports to Free-BSD and Darwin.

We are currently working on version 2.0, a development release is available, but it is not well tested. It will probably take a few months before version 2.0 sees a public release.

A new paper about buddha will appear in the forthcoming proceedings of the Advanced Functional Programming Summer School, which was held in Tartu, August 2004 (→ 1.5.1).

Further reading

www.cs.mu.oz.au/~bjpop/buddha

5.4.4 QuickCheck

Report by:	Koen Claessen and John Hughes
Status:	active development

QuickCheck is a tool for specifying and testing formal properties of Haskell programs. There have been several inofficial draft versions of QuickCheck around.

Right now we are in the process of packaging up a new, official version of QuickCheck, integrating support for:

- automatic finding of small counter examples
- monadic properties
- exception handling amd time-outs
- stating properties that are expected to fail
- a callback hook for displaying failing test cases
- generating test reports

And lots lots more! We plan to distribute the new QuickCheck using the new Haskell Cabal (→ 4.1.1).

An accompanying tutorial, explaining typical problems and programming idioms that solve them is also in the make.

5.4.5 HUnit

There have been no recent functional changes to HUnit. However, Malcolm Wallace recently imported HUnit into the fptools CVS repository, adjusted the module names to fit the hierarchical scheme, and linked HUnit to the package build system for nhc98. We intend that HUnit also appear with GHC and Hugs before long.

Further reading

http://hunit.sourceforge.net/

5.5 Development

5.5.1 hmake

Report by:	Malcolm Wallace
Status:	stable, maintained

Hmake is an intelligent compilation management tool for Haskell programs. It is stable at public release version 3.08, with occasional maintenance and bugfixes to the CVS tree at haskell.org.

Further reading

http://haskell.org/hmake

5.5.2 cpphs

Report by:	Malcolm Wallace
Status:	active development

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

Further reading

http://haskell.org/cpphs

5.5.3 Visual Studio support for Haskell

Report by:	Simon Marlow
Status:	in development

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. So far we have various editor features completed:

- syntax colouring
- continuous indication of parse errors, static errors and type errors as you edit Haskell code.
- Jump to the definition of an identifier (current module only for now).
- Hover the mouse over an identifier to show its type

Support for projects is now maturing: basic project support is working, and integrated with Cabal. Visual Haskell can load Cabal packages, and projects created in Visual Haskell are fully-fledged Cabal packages themselves, and can be built & installed on machines without Visual Haskell.

The current release plan is to put out an initial release coinciding with GHC 6.4, which should be around the end of 2004.

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 Studio/Haskell, please drop me a note (Simon Marlow ⟨simonmar@microsoft.com⟩).

5.5.4 Haskell support for the Eclipse IDE

Report by:	Leif Frenzel
Status:	working, though alpha

The Eclipse platform is an extremely extensible framework for IDEs (implemented in Java), developed by an Open Source Project. This project extends it with tools to support Haskell development.

The aim is to develop an IDE for Haskell that provides the set of features and the user experience known from the Eclipse Java IDE (the flagship of the Eclipse project), and integrates a broad range of compilers, interpreters, debuggers, documentation generators and other Haskell development tools. Long-term goals include a language model with support for intentional programming, refactoring and structural search.

The current version is 0.5 (considered 'alpha'). It features a project model, a configurable source code editor (with syntax coloring and Code Assist), compiler support for GHC and launching from the IDE. Since the last HC&A Report we have improved support for GHC options and included facilities for running Hugs, GHCi and Haddock (→ 5.5.5) from the IDE. The goal for the next months is to make the IDE more language-aware.

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

Further reading

- http://eclipse.org
- http://lists.sourceforge.net/lists/listinfo/eclipsefp-develop
- Project homepage: http://eclipsefp.sf.net

5.5.5 Haddock

Report by:	Simon Marlow
Status:	stable, maintained

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

I've recently been experimenting with adding support for "collapsible sections" to the HTML output. For example, the instances of a type or class would be hidden by default, and could be expanded by clicking a button. Provided this can be made to work reliably across the browsers that Haddock currently supports, it will be in the next release.

Further reading

- 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/

6 Applications

6.1 Non-commercial applications

6.1.1 HScheme

Report by:	Ashley Yakeley

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.

Very little work is currently being done on it though, as the developer's free time has been shortened by gainful employment. Further work may resume, at a reduced pace, once left-over issues in the latest JVM-Bridge (→ 5.1.3) have been cleared up.

Further reading

http://hscheme.sourceforge.net/

6.1.2 Curryspondence

Report by:	Shae Erisson

Curryspondence is a mailing list searching program using HaskellDB (→ 4.4.5) and WASH (→ 4.7.4). At the moment it takes input in the form of mailman mbox files to populate a postgresql database, but it should work with any supported HaskellDB backend. The demo is down for the moment, should be back soon at http://www.ScannedInAvian.org/cgi-bin/curryspondence/ darcs repo: http://www.ScannedInAvian.org/repos/curryspondence

6.1.3 lambdabot

Report by:	Shae Erisson

lambdabot is an IRC robot with a simple plugin architecture. Plugins include a lambda calculus interpreter, dictd client, fortune cookie, and more. You can download lambdabot from the darcs repo here:

```
darcs get
  http://www.scannedinavian.org/repos/lambdabot
```

6.1.4 HWS-WP

Report by:	Simon Foster

The Haskell Web-Server With Plugins (HWS-WP) is a simple HTTP server written in Haskell, originally implemented by Simon Marlow and then updated by Martin Sjögren, who implemented a simple plug-in system for handling requests. After some work, HWS-WP has been resurrected (it was initially not working with GHC 6+) and now can be used to serve out simple websites. It has also been used as a base on which to demonstrate HAIFA's Application Container.

HWS-WP still requires much work, notably it needs strengthening and its implementation of HTTP needs bringing up to compatibility with the RFCs. A better plug-in system with multiple module support and dependency tracking is also proposed. The current version of HWS-WP can be obtained from http://sourceforge.net/projects/haskell-libs/

6.1.5 Hircules, an irc client

Report by:	Jens Petersen

Hircules is a gtk2-based IRC client built on gtk2hs and code from lambdabot. The last release is still version 0.3, though I have various bug fixes and improvements that I hope to release soon. New features in 0.4 will include basic text search and improved channel nicks handling.

Further reading

http://haskell.org/hircules/

6.1.6 Darcs

Report by:	David Roundy
Status:	active development

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 version 1.0.0 was released on November 8, 2004 after considerable improvement in the past six months, including performance enhancements, bug fixes and improvements in the user interface. Right now the new features for the next major release are in the process of being planned, and this is a great time to become involved, if you're looking for a Haskell project to join.

Darcs is free software licensed under the GNU GPL.

Further reading

http://darcs.net

6.1.7 Yarrow

Report by: Frank Rosemeier

From the Yarrow web pages:

"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 P. Barendregt: Lambda Calculi with Types; in D.M. Gabbai, S. Abramsky, and T.S.E. Maibaum (editors): Handbook of Logic in Computer Science, volume 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."

In 2003 Frank Rosemeier has ported Yarrow (written by Jan Zwanenburg using Haskell 1.3, see http://www.cs.kun.nl/~janz/yarrow/) to Haskell 98. The Haskell 98 source code has been published on his *old homepage* near http://www.fernuni-hagen.de/MATHEMATIK/ALGGEO/Mitarbeiter/. This year a *new homepage* will be put somewhere near link "Mitarbeiter" of http://www.informatik.fernuni-hagen.de/import/thi1/. A new *Yarrow homepage* located at http://www.haskell.org/yarrow/ will contain a copy of the homepage for the Haskell 1.3 version as well as the Haskell 98 adaption.

6.1.8 HasLaTeX

Report by: Frank Rosemeier

This year Frank Rosemeier has begun to write some Haskell 98 code of a LaTeX translator (for LaTeX see http://www.latex-project.org/). The system shall parse LaTeX 2_ε documents and convert them to other formats, e.g. into plain ASCII-Text. The idea is to provide a Haskell library (called HasLaTeX) for parsing and digesting LaTeX files (using Parsec (→ 4.3.1) and probably PPrint), which may be useful for other applications.

The development of this project has been temporarily postponed due to a change in the job of the author.

6.1.9 DoCon, the Algebraic Domain Constructor

Report by: Serge Mechveliani

DoCon is a program for *symbolic computation in mathematics*, written in Haskell. It is a package of modules distributed freely, with the source program and manual.

DoCon joins the *categorial approach* to the mathematical computation expressed via the Haskell type classes, and explicit processing of the *domain description terms*. It implements a good piece of commutative algebra: linear algebra, polynomial gcd, factorization, Groebner bases, and other functions. They are programmed under the very *generic assumptions*, like "over any Euclidean ring", over any GCD-ring, any field, and so on. DoCon also supports the *constructions on domains*: Fraction, Polynomial, Residue ring, and others. That is certain set of operations on a constructed domain is built automatically.

DoCon is written in what we call Haskell-2-pre – certain functional extension of Haskell-98. This extension includes the multiparametric classes, overlapping instances, other minor features.

My intention for 2004 is as follows.

- To release DoCon-2.08 (the Algebraic Domain Constructor) running under ghc-6.2.2. DoCon-2.08 is ready, it waits for the official and reliable ghc-6.2.2 release to appear.

- To release in the end of 2004 the prover Dumatel-1.02 based on Term Rewriting, Completion, and inductive reasoning.

6.1.10 lhs2TEX

Report by:	Andres Löh
Status:	stable, maintained

This tool by Ralf Hinze and Andres Löh is a preprocessor that transforms literate Haskell code into LaTeX documents. The output is highly customizable by means of formatting directives that are interpreted by lhs2TEX. Other directives allow the selective inclusion of program fragments, so that multiple versions of a program and/or document can be produced from a common source. The input is parsed using a liberal parser that can interpret many languages with a Haskell-like syntax, and does not restrict the user to Haskell 98.

The program is stable and can take on large documents: it handles my complete Ph.D. thesis without any problems, and I see that Graham Hutton makes use of lhs2TEX in his new book (→ 1.4.1).

Since the last report, I have made an experimental Windows installer, added a few minor things, and started to collect a few library files with often-used formatting directives, such as to allow code on colored backgrounds or line numbering for code blocks.

These changes are currently only available if you checkout the development version from the Subversion repository, but I hope to make a release sometime soon.

Further reading

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

6.1.11 Audio signal processing

Report by:	Henning Thielemann
Status:	experimental, active development

In this project audio signals are processed using pure Haskell code. This includes a simple signal synthesis backend for Haskore, filter networks, signal processing supported by physical units.

Future plans

Connect with the HaskellDSP library. Hope on faster code generated by some Haskell compilers. :-) Probably connect to some software synthesizer which is more efficient, but nearly as flexible as code entirely written in Haskell. Explore whether Monads and Arrows can be used for a more convenient structuring and notation of signal algorithms.

Further reading

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

6.1.12 Converting knowledge-bases with Haskell

Report by:	Sven Moritz Hallberg

I work in a research project concerned with knowledge-based configuration. We need to convert XML knowledge-bases from a commercial tool (EngCon) to the LISP-based description language understood by our in-house tool (Konwerk). While the knowledge representations are similar in many ways, they slightly differ in structure. The converter, implemented in Haskell, reads the XML into a first internal data structure, converts that to a slightly different internal structure which can then be output in the desired form. Haskell's data type facilities made it easy to abstract the data from its concrete external representation. HaXML (→ 4.7.2) provided for reliable XML parsing. Parsec (→ 4.3.1) was also of great value, allowing for fabulously easy building of an expression parser. In the end, Haskell allows us to stick to LISP when everyone else is using Java. :)

6.1.13 NetEdit

Report by:	Daan Leijen

NetEdit is a graphical editor for Bayesian networks that is developed by the Decision Support System group of Utrecht University. It is written in Haskell and uses wxHaskell (→ 4.5.2) as its GUI library. For inference it uses the C++ library SMILE developed by the Decision Systems Laboratory of Pittsburgh University. Features (will) include test selection, logic sampling, sensitivity analysis and qualitative networks. The application runs on both Windows and Linux. Screenshots can be found on the wxHaskell webpage: http://wxhaskell.sourceforge.net

6.1.14 riot

Report by:	Tuomo Valkonen

Riot is a tool for keeping (textual) information organised. Some people call such programs 'outliners'. It is a todo list and note manager, and a manager for whatever information one might collect. Riot has a curses-based interface resembling those of slrn and mutt and all text editing is done with your favourite external editor: Riot is just a nice-to-use browser and entry organiser for collections of text. The Riot homepage is at http://iki.fi/tuomov/riot/.

6.1.15 Flippi

Report by: Philippa Cowderoy

Flippi is a lightweight (and currently somewhat underfeatured) wiki clone written in Haskell and released under the BSD license. The current release is v0.03, which added support for scripting and a RecentChanges script. The main planned feature for the next release is a template facility, with various interface alterations in the default setup being likely. Also in the pipeline is a refactoring of the parser to make adding new pieces of markup syntax easier, and metadata support and revision histories with reversion are planned by v0.1.

A goal in development so far, and one which the author would like to maintain, is to keep the code easy to understand and modify – to this end, the configuration is currently all done by source modification. This isn't necessarily as bad as it sounds – if the Flippi CGI is run via runhugs or similar, there's no perceivable difference to somebody configuring Flippi bar the level of power available. However, so far Flippi has only been tested under GHC 6.2 and is dependant on a recent version of the hierarchical libraries.

Further reading

- http://www.flippac.org/projects/flippi/
- http://www.scannedinavian.org/cgi-bin/flippi/flippi

6.1.16 Postmaster ESMTP Server

Report by: Peter Simons

Postmaster is an Internet mail transport agent (MTA) written and configured in Haskell. At the time of this writing, it handles incoming ESMTP network connections and delivers accepted messages to the user's mailbox by piping it into an arbitrary local mailer (e.g. Procmail).

As is to be expected from an MTA written in Haskell, it is configurable beyond anything you'll ever need. The server itself comes as a monadic combinator library; so you can plug together or modify the components as you please. A pretty sophisticated standard configuration on which to build is part of the distribution.

Postmaster is still very young; there remains a lot to be done before it can really compete with Sendmail or Postfix. Most notably, it lacks any form of queue management right now. Nonetheless, for leaf sites, which don't need to do extensive mail relaying, it is a reliable and powerful solution already.

Further details are available at: http://postmaster.cryp.to/

It is worth noting that Postmaster includes several generally useful libraries which are not tied to the ESMTP server:

BlockIO implements a monad for fast, non-blocking I/O with static `Ptr Word8` buffers.

HsDNS implements an asynchronous DNS resolver an top of the GNU adns library.

HsEMail Parsec (→ 4.3.1) parsers for most of RFC 2821 and 2822.

Child provides `spawn`, `par`, and `timeout` for more flexible handling of child computations started with `forkIO`.

Syslog FFI (→ 3.1) bindings to the `syslog(3)` system API.

hOpenSSL (very incomplete) FFI bindings to the OpenSSL library. At the moment provides mostly access to the `libcrypto` part.

6.1.17 yi

Report by: Donald Bruce Stewart
Status: just started, active development

yi is a project to write a Haskell-extensible editor. The project is still in its infancy, however much usable code exists. yi is structured around an basic editor core, such that most components of the editor can be overridden by the user, using configuration files written in Haskell. Current work is focused on producing full vi-compatibility for version 0.1.

The source repository is available:

```
darcs gethttp://www.cse.unsw.edu.au/~dons/yi
```

6.2 Commercial users

6.2.1 Reid Consulting Ltd

Report by: Alastair Reid

Reid Consulting (UK) Ltd is closing down after 2.75 profitable years. RCL's main contracts over that time involved writing compiler tools in Haskell, writing a little language for configuring real time systems (also in Haskell), writing an abstract interpreter for machine code (in C, for historical reasons), and generating optimal abstract transfer functions for abstract interpreters (in C++). RCL is closing because Alastair Reid has taken a job with ARM Ltd in Cambridge.

6.2.2 Galois Connections, Inc.

Report by: Andy Moran

Galois (aka Galois Connections, Inc.) is an employee-owned software development company based in Beaverton, Oregon, U.S.A. John Launchbury, Andy Gill, Jeff Lewis, and Andy Moran started Galois in late 1999 with the stated purpose of using functional languages to solve industrial problems.

Galois develops software under contract, and every project (bar two) that we have ever done has used Haskell; the exceptions used SML-NJ and OCaml, respectively. We've delivered tools, written in Haskell, to clients in industry and the U.S. government that are being used heavily. Four diverse examples: Cryptol, a domain-specific language for cryptography (with an interpreter and a compiler); a GUI debugger for a specialized chip; a tool for easily embedding new syntax in the client's own language (sort of a souped-up Happy + OCaml's P4), and a legacy code translator (translating from K&R C to ANSI C, while moving from SunOS 4 to Solaris and a new abstract API).

So, why do we use Haskell? There are benefits to moving to Java or C# from C++ or C, such as cleaner type systems, cleaner semantics, and better memory management support. But languages like Haskell give you a lot more besides: they're much higher level, so you get more productivity, you can express more complex algorithms, you can program and debug at the "design" level, and you get a lot more help from the type system. These arguments have been made time and again though, and they're also pretty subjective.

For Galois, it's also a big bonus that Haskell is close to its mathematical roots, because our clients care about "high assurance" software. High assurance software development is about giving solid (formal or semi-formal) evidence that your product does what it should do. The more functionality provided, the more difficult this gets. The standard approach has been to cut out functionality to make high assurance development possible. But our clients want high assurance tools and products with very complex functionality. Without Haskell (or some similar language), we wouldn't even be able to attempt to build such tools and products.

At Galois, we're happily able to solve real world problems for real clients without having to give up on using the tools and languages we worked on when we were in the Academic world. In fact, we credit most of our success with the fact that we can apply language design and semantics techniques to our clients' problems. Functional languages are an integral part that approach, and a big part of the unique value that our clients have come to known us for.

The good news is that our business is working quite well. As of fall 2004, Galois is 16 engineers strong, with a support staff of 5. Our revenues have doubled in the past 2 years, and we have been profitable each quarter over that same period. In fact our biggest challenge is keeping up with the demand for our services!

In short: Hard Problems + FP == Fun + Profit!

Further reading

http://www.galois.com/.

6.2.3 Aetion Technologies LLC

Report by: Mark Carroll

Aetion Technologies LLC continues to use Haskell for most of its in-house development.

Aetion's principal source of revenue is from prototyping applications of our artificial intelligence techniques for the US Department of Defense. We are also researching applications for risk management in financial decision making. Aetion is in the process of releasing to the open-source community work that we have done in Haskell that was necessary for our products but incidental to our core competitive advantages.

Further reading

http://www.aetion.com/

6.3 Haskell in Education

6.3.1 Haskell in Education at Universidade de Minho

Report by: Jorge Sousa Pinto

Haskell is heavily used in the undergraduate curricula at Minho. Both Computer Science and Systems Engineering students are taught two Programming courses with Haskell. Both programmes of studies fit the "functional-first" approach; the first course is thus a classic introduction to programming with Haskell, covering material up to inductive datatypes and basic monadic input/output. It is taught to 200 freshmen every year. The second course, taught in the second year (when students have already been exposed to other programming paradigms), focuses on pointfree combinators, inductive recursion patterns, functors and monads; rudiments of program calculation are also covered. A Haskell-based course on grammars and parsing is taught in the third year, where the HaLeX library is used to support the classes.

Additionally, in the Computer Science curriculum Haskell is used in a number of other courses covering Logic, Language Theory, and Semantics, both for illustrating concepts, and for programming assignments.

Minho's 4th year course on Formal Methods (a 20 year-old course in the VDM tradition) is currently being restructured to integrate a system modeling tool based on Haskell and VooDooM. Finally, in the last academic year we ran an optional, project-oriented course on Advanced Functional Programming. Material covered here focusses mostly on existing libraries and tools for Haskell, such as YAMPA - functional reactive programming with arrows, the WASH library, the MAG system, the Strafunski library, etc. This course benefitted from visits by a number of well-known researchers in the field, including Ralf Laemmel and Peter Thiemann.

7 Groups

7.1 Research Groups

7.1.1 Artificial Intelligence and Software Technology at JWG-University Frankfurt

Report by:	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 using the FUNDIO calculus as semantics for the GHC core 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. A detailed description of our investigation is available as a technical report from the DIAMOND project page. By turning off the few transformations which are not FUNDIO-correct and those that have not yet been investigated (especially most of the global ones), we have achieved a FUNDIO-compatible modification of GHC which is called *HasFuse*.

HasFuse correctly compiles Haskell programs which make use of 'unsafePerformIO' in the common (safe) sense, since the problematic optimizations that are mentioned in the documentation of the System.IO.Unsafe module (let floating out, common subexpression elimination, inlining) are turned off or performed more restrictively. But HasFuse also compiles Haskell programs which make use of 'unsafePerformIO' in arbitrary contexts. Since the call-by-need semantics of FUNDIO does not prescribe any sequence of the IO operations, the behavior of 'unsafePerformIO' is no longer 'unsafe'. I.e. 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 IO operations will not astonishingly be duplicated.

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

Non-deterministic call-by-need lambda calculi

Important topics are to investigate static analyses based on the operational semantics, to obtain more inference rules for equality in call-by-need lambda-calculi, e.g. a definition of behavioural equivalence. *Matthias Mann* has established a proof of its soundness w.r.t. contextual equivalence for a non-deterministic call-by-need lambda calculus. Further research is aimed towards extensions of this calculus to support work on strictness analysis using abstract reduction.

A new result is a correctness proof of this algorithm, which has been implemented at least twice: Once by Nöcker in C for Concurrent Clean and on the other hand by Schütz in Haskell in 1994. A technical report covering the latter is available from our website. The proof of correctness of strictness analysis using abstract reduction uses a conjecture that the defined behavioural equivalence is included in the contextual equivalence.

Implementations Using Haskell

As a final year project, *Christopher Stamm* implemented an 'Interpreter for Reduction Systems' (*IfRS*) in Haskell. IfRS is an interpreter for higher order rewrite systems that are based on structural operational semantics. Additionally, it is possible to define reduction contexts and to use contexts and domains (term sets that are definined similiar to contexts without holes) in the rewrite rules. Also, IfRS is able to test whether the reduction rules satisfy the conditions

of the GDSOS-rule format. The GDSOS-rule format ensures that bisimulation is a congruence.

Current research topics of our group also encompass second order unification, higher order unification and context unification. It is an open problem whether (general) context unification is decidable. *Jörn Gersdorf* has implemented a non-deterministic decision algorithm for context matching in Haskell which benefits from lazy evaluation at several places.

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
- IfRS – Interpreter for Reduction Systems
  http://www.informatik.uni-frankfurt.de/~stamm

7.1.2 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 project and the Common Algebraic Specification Language (CASL).

The UniForM workbench is a tool integration framework mainly geared towards tools for formal methods. During the MMiSS project, it has been extended to a repository providing configuration management, version control and change management for semantically structured documents. The UniForM workbench and MMiSS repository currently contain over 100k lines of Haskell code.

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

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, and Template Haskell. Further tools actively used are DriFT (→ 3.4), Haddock (→ 5.5.5), the combinator library Parsec (→ 4.3.1), and Hatchet.

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
- 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

7.1.3 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⟩.

7.1.4 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. Work has progressed on a number of the following projects:

- We are looking at using our port of Haskell (→ 2.6.1) for embedded DSLs to build handheld applications.
- Kate Krastev is investigating specialization of the nhc98 runtime with a view to code compaction.
- Phuong Tri is working on program proving for Haskell using Isabelle.

- Qingsong Ye and a number of other students are looking at designing embedded DSLs for specifying different aspects of handheld applications, including data synchronization and user interface.
- We are also interested in using Haskell or similar languages as the basis for language processor specification, so we are looking at topics such as parser combinators and first-class attribute grammars.

Unfortunately, none of these projects is ready for a public release at the moment.

Further reading

Our new website is slowly being populated with information on all of our projects: http://www.comp.mq.edu.au/plrg/

In the meantime, please contact us via email to ⟨plrg@ics.mq.edu.au⟩.

7.1.5 Functional Programming at the University of Kent

Report by: Olaf Chitil

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 continuously improving the visual interactive programming system Vital (→ 2.6.4). Axel Simon keeps maintaining the gtk2hs binding to the Gtk+ GUI library (→ 4.5.5) and had been involved in coordinating the Haskell GUI efforts (→ 4.5.1). Chris Ryder is improving his Metrics and Visualization library Medina (→ 4.3.3). Huiqing Li, Simon Thompson and Claus Reinke have released further snapshots of HaRe, the Haskell Refactorer (→ 5.3.4). Olaf Chitil and Thomas Davie continue together with the York group extending and improving the Haskell tracer Hat (→ 5.4.2).

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/
- MEDINA: http://www.cs.kent.ac.uk/~cr24/medina/
- Refactoring Functional Programs: http://www.cs.kent.ac.uk/projects/refactor-fp/
- Hat: http://www.haskell.org/hat/

7.1.6 Parallel and Distributed Functional Languages Research Group at Heriot-Watt University

Report by: Phil Trinder
Members: Abyd Al Zain, Andre Rauber Du Bois, Gudmund Grov, Robert Pointon, Greg Michaelson, Phil Trinder, Jan Henry Nyström, Chunxu Liu, Graeme McHale, Xiao Yan Deng

The Parallel and Distributed Functional Languages (PDF) research group is part of the Dependable Systems Group in Computer Science at the School of Mathematics and Computer Science at Heriot-Watt University.

The group investigates the design, implementation and evaluation of high-level programming languages for high-performance, distributed and mobile computation. The group aims to produce notations with powerful yet high-level coordination abstractions, supported by effective implementations that enable the construction of large high-performance, distributed and mobile systems. The notations must have simple semantics and formalisms at an appropriate level of abstraction to facilitate reasoning about the coordination in real distributed/mobile systems i.e. to transform, demonstrate equivalence, or analyze the coordination properties. In summary, the challenge is to bridge the gap between distributed/mobile theories, like the pi and ambient calculi, and practice, like CORBA and the OGSA.

Languages

The group has designed, implemented, evaluated and used several high performance/distributed functional languages, and continues to do so. High performance languages include Glasgow parallel Haskell (→ 3.2.1) and Parallel ML with skeletons (PMLS). Distributed/mobile languages include Glasgow distributed Haskell (→ 3.2.2), Erlang (http://www.erlang.org/), Hume (http://www-fp.dcs.st-and.ac.uk/hume/), JoCaml and Camelot.

Collaborations

Primary industrial collaborators include groups in Microsoft Research Labs (Cambridge), Motorola UK Research labs (Basingstoke), Ericsson, Agilent Technologies (South Queensferry).

Primary academic collaborators include groups in Complutense Madrid, JAIST, LMU Munich, Phillips Universität Marburg, and St Andrews.

Further reading

http://www.macs.hw.ac.uk/~ceeatia/PDF/

7.1.7 Programming Languages & Systems at UNSW

Report by: Manuel Chakravarty

The PLS research group at the University of New South Wales has produced the C–>Haskell (→ 5.1.2) interface generator and more recently the hs-plugins (→ 4.2.12) library for dynamically loaded type-safe plugins. As a testbed for further research in dynamic code loading and type checking, we have just started a project developing a highly customisable editor in Haskell. We also recently released PanTHeon, a portable re-implementation of Conal Elliott's Pan animation tool based on meta-programming in Template Haskell.

We are also interested in bringing some of the benefits of functional intermediate languages to the world of compilers for imperative languages. After some initial work on using Administrative Normal Form (ANF), a variant of the lambda calculus, for implementing optimisations originally formulated for the imperative Static Single Assignment (SSA) form, we are now working at an optimising compiler for the language C that is based on ANF and implemented in Haskell.

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

7.1.8 Logic and Formal Methods group at the Informatics Department of the University of Minho, Braga, Portugal

Report by: Jorge Sousa Pinto

We are a group of about 12 staff members and various PhD and MSc students. We have shared interest in formal methods and their application in areas such as data and code reverse and re-engineering, program understanding, and communication protocols. Haskell is our common language for teaching and research.

Haskell is used as first language in our graduate computers science education (→ 6.3.1). José Valença and José Barros are the authors of the first (and only) Portuguese book about Haskell, entitled "Fundamentos da Computação" (ISBN 972-674-318-4). Alcino Cunha has developed the Pointless library for point-free programming in Haskell (→ 4.2.11), as well as the DrHylo tool that transforms functions using explicit recursion into hylomorphisms. Supervised by José Nuno Oliveira, students Tiago Alves and Paulo Silva are developing the VooDooM tool (→ 5.3.5), which transforms VDM datatype specifications into SQL datamodels and students João Ferreira and José Proença will soon start developing CPrelude.hs, a formal specification modelling tool generating Haskell from VDM-SL and CAMILA. João Saraiva is responsible for the implementation of the attribute system LRC, which generates (circular) Haskell programs. He is also the author of the HaLex library and tool, which supports lexical analysis with Haskell. Joost Visser has developed Sdf2Haskell, which generates GLR parsing and customizable pretty-printing support from SDF grammars, and which is distributed as part of the Strafunski bundle. Most tools and library modules develop by the group are organized in a single infrastructure, to facilitate reuse, which can be obtained as a single distribution under the name UMinho Haskell Libraries and Tools.

The group has recently started the 3-year project called PURe which aims to apply formal methods to Program Understanding and Reverse Engineering. Haskell is used as implementation language, and various subprojects have been initiated, including Generic Program Slicing.

Further reading

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

7.1.9 The Computer Systems Design Laboratory at the University of Kansas

Report by: Perry Alexander

The Computer Systems Design Laboratory at the University of Kansas is using Haskell in several distinct projects.

We are continuing work, previously reported in the Communities and Activities report, developing tools for the Rosetta specification language. This work uses Haskell as the primary platform for a toolset facilitating the analysis of heterogeneous models written in Rosetta. We use a composable interpreter framework to provide basic language interpretation and a collection of static and dynamic analysis tools.

A related project utilizes Haskell in the development of a generalized proof assistant, Prufrock. This provides a framework for integrating new languages into the proof environment. The language representation is separated from the logical inference rules, using generic programming techniques. Proof tactics (written in Haskell), interaction, and specific prover implementation, including such features as global state and logging are separated using Haskell's type class system. This results in a set of (largely) independent modules that can be combined to produce a specialized first-order theorem prover for a given language and a given system of inference. A technical report, including the entire Prufrock source, is available at the Prufrock website.

Another project explores the implementation of functional languages, via graph reduction, on FPGA hardware. This system combines a compiler and simulator,

written exclusively in Haskell, with a VHDL implementation of the abstract machine. A graph reduction machine is being synthesized in FPGA from VHDL source to directly execute compiled Haskell.

Finally, we are Haskell to development a formalism to represent dance. Currently, choreographers communicate only by performance. The dance language aims to provide a mechanism for to allow choreographers a means to communicate the structure of a routine textually. Additionally, the dance language has an associated typechecker, used by choreographers to detect errors in the transcription of routines.

Further reading

- Computer Systems Design Lab: http://www.ittc.ku.edu/research/view_lab.phtml?lab=CSDL
- Systems Level Design Group: http://www.ittc.ku.edu/Projects/SLDG/
- Rosetta Specification Language: http://www.sldl.org
- Prufrock: http://www.ittc.ku.edu/~wardj/prufrock/
- Dance Language: http://www.ittc.ku.edu/~jenis/

7.1.10 Cover: Combining Verification Methods

Report by:	Patrik Jansson
Participants:	John Hughes, Thierry Coquand, Peter Dybjer, Mary Sheeran, Marcin Benke, Koen Claessen, Patrik Jansson, Andreas Abel, Gregoíre Hamon, Ulf Norell, Fredrik Lindström, Nils Anders Danielsson

Cover is a Haskell-centered research project at Chalmers funded by the Swedish Foundation for Strategic Research. The goal is to develop methods for improving software quality. The approach is to integrate a variety of verification methods into a framework which permits a smooth progression from hacking code to fully formal proofs of correctness.

More concretely we work on these components:

- QuickCheck – automated random testing (→ 5.4.4)
- Agda – a dependently typed language and its proof engine (implemented in Haskell) (→ 2.6.6)
- Cover Translator – translation from Haskell to
  - Agda – for interactive proof
  - First order logic – for automated proof

Our best results so far include:

- Development of QuickCheck (automatic shrinking, monadic testing, etc.)
- Development of Agda (built-in types, a class system, "implicit arguments", etc). Development of methodology for reasoning about general recursive programs.
- Cover Translator. Translates Haskell (via GHC Core) into a first order formulas understood by automatic theorem provers such as Gandalf and Vampire. Ongoing work on case studies.
- A first prototype - an extension of Alfa (an advanced GUI for the prover Agda) with tools for testing and automatic proof construction.
- Agsy – automatic proof search plugin for Agda.
- Collaboration with AIST (Advanced Industrial Science and Technology Institute in Japan) on development and applications of Agda.

The Cover source code can be browsed at http://cvs.coverproject.org/marcin/cgi/viewcvs/ and can be accessed by anonymous CVS from cvs.coverproject.org.

Short term goals:

- Complete the chain of tools (QuickCheck, Agda, Cover Translator) so that we can take actual Haskell code, test, translate for the theorem provers, and prove properties.
- Identify larger scale case studies that ought to be tractable for our methods.

Further reading

For more details about the project, read about QuickCheck (→ 5.4.4) and Agda (→ 2.6.6) in this report or consult the homepage at http://coverproject.org.

7.2 Other groups

7.2.1 Debian Users

Report by:	Isaac Jones

The Debian Haskell community continues to grow, with both new users and developers appearing. Together with work on cabal and libraries (→ 4.1.1) we are working towards providing a much improved Haskell development environment, and the number of applications in Debian written in Haskell is also continuing to grow. A summary of the current state can be found on the Haskell Wiki (→ 1.3): http://www.haskell.org/hawiki/DebianUsers.

For developers, we have a prototype policy for packaging tools for Debian: http://urchin.earth.li/~ian/haskell-policy/haskell-policy.html/.

For users and developers, we have also started a mailing list: http://urchin.earth.li/mailman/listinfo/debian-haskell.

In order to provide backports, bleeding edge versions of Haskell tools, and a place for experimentation with packaging ideas, Isaac Jones

and Ian Lynagh have started the "Haskell Unsafe" Debian archive (http://haskell-unsafe.alioth.debian.org/haskell-unsafe.html) where a wide variety of packages can be found. This was recently moved to a Debian server.

7.2.2 Haskell packages for Fedora Core

Report by: Jens Petersen

Yum and apt repositories for Haskell rpm packages have been setup. At the time of writing there are packages for c2hs-0.13.1, cabal-0.1, darcs-0.9.22-1, ghc-6.2.1, greencard-3.01, gtk2hs-0.9.5.50, hircules-0.3 and hs-plugins-0.9.6. More to come, including updating to ghc-6.2.2. Contributions are much welcome. And I would still like to see ghc added to Fedora Extras.

Further reading

http://haskell.org/fedora/

7.2.3 OpenBSD Haskell

Report by: Donald Bruce Stewart

Haskell support on OpenBSD has continued to improve over the last 6 months. A page documenting the current status of Haskell on OpenBSD is at http://www.cse.unsw.edu.au/~dons/haskell_openbsd.html.

GHC is now fully supported on the i386, including ghci. GHC is also distributed for the amd64 and sparc. nhc98-1.16 is available for i386, sparc and powerpc. Hugs is available for the alpha, amd64, hppa, i386, powerpc, sparc and sparc64.

7.2.4 Haskell in Gentoo Linux

Report by: Andres Löh

The support for Haskell and Haskell-related packages in Gentoo Linux is improving slowly, but steadily. Recently a call for new developers was sent out to the Gentoo mailing lists and in the Gentoo newsletter. Although there have been plenty of reactions, new help and, most of all feedback, is always welcome.

Next to adding additional packages, there currently are the following longer-term projects:

- move GHC package management calls to an eclass which many Haskell ebuilds can use, port the existing ebuilds to use the eclass, and improve the eclass to support Cabal (→ 4.1.1);
- add a `ghc-updater` script that facilitates rebuilding of Haskell libraries on a GHC upgrade;
- make Haskell available on as many Gentoo-supported platforms as possible.

New ebuilds, comments and suggestions, and bug reports can be filed at bugs.gentoo.org. Make sure that you mention "Haskell" in the subject of the report.

8 Individual Haskellers

8.1 Oleg's Mini tutorials and assorted small projects

Report by: Oleg Kiselyov

The page about type system hacks (http://pobox.com/~oleg/ftp/Haskell/types.html) – a part of the collection of various Haskell mini-tutorials and assorted small projects (http://pobox.com/~oleg/ftp/Haskell/) – has received two additions:

Functions with the variable number of (variously typed) arguments

It is sometimes claimed that Haskell does not have polyvariadic functions. Here we demonstrate how to define functions with indefinitely many arguments; those arguments do not have to be of the same type. The code shows that defining polyvariadic functions takes only a few lines of Haskell code, and requires only the most common extension of multiparameter classes with functional dependencies. We give the complete description of the technique, the explanation of the type inference for functions with the variable number of arguments, and many examples.

Partial signatures

The regular (full) signature of a function specifies the type of the function and enumerates all of the applicable typeclass constraints. The list of the constraints may be quite large. Partial signatures help when: (i) we wish to add an extra constraint to the type of the function but we do not wish to explicitly write the type of the function and enumerate all of the typeclass constraints, (ii) we wish to specify the type of the function and perhaps some of the constraints – and let the typechecker figure out the rest of them. Contrary to a popular belief, both of the above are easily possible, in Haskell 98.

8.2 Graham Klyne

Report by: Graham Klyne

My primary interest is in RDF http://www.w3.org/RDF/ and Semantic Web http://www.w3.org/2001/sw/ technologies. Since my submission for the March 2004 HC&A Report, I have updated my Swish package http://www.ninebynine.org/RDFNotes/Swish/Intro.html to include a graph-differencing facility, and RDF input from CSV files (e.g. Excel spreadsheet export). I have also restructured the code and implemented a number of small refinements but these changes are not yet in a formally released version of the software, though a working copy can be found on my web site http://www.ninebynine.org/Software/HaskellRDF/RDF/, http://www.ninebynine.org/Software/HaskellUtils/.

My implementation of a replacement for the Network.URI module has been updated to reflect some clarifications in the work-in-progress revised URI specification http://gbiv.com/protocols/uri/rev-2002/rfc2396bis.html. This software can be found at http://www.ninebynine.org/Software/HaskellUtils/Network/. I believe this is now ready for incorporation into the Haskell common library CVS repository. As well as a simple stand-alone test suite, there is also an RDF-based test suite for the URI handling code, which uses my Swish code, which allows the test cases to be updated by generating a new RDF/CSV description file.

I have created a copy of HaXml (→ 4.7.2) that is extensively modified to be more conformant with the W3C XML test suite. This has support for UTF-8 and UTF-16 character encoding, external entities with HTTP URIs (using a lightly-modified copy of Björn Bringert's version of HTTP modules), XML namespaces, xml:lang and xml:base directives. I believe this now contains much of the XML parsing functionality needed for real XML-based Web applications. The software (with test suite) is available at http://www.ninebynine.org/Software/HaskellUtils/HaXml-1.12/ and my copy of the HTTP code is at http://www.ninebynine.org/Software/HaskellUtils/Network/.

I have also implemented a full RDF/XML parser that passes all official test cases (and some others) except those dealing with Unicode character normalization http://www.ninebynine.org/Software/HaskellRDF/RDF/Harp/. The plan is to integrate this into my Swish package, but that has not yet been done.

I am currently experimenting with an implementation of description logic http://dl.kr.org/ reasoners in literate Haskell. This is intended primarily as a self-tutorial exercise, and so far I've completed a simple structural subsumption reasoner, but I eventually hope to integrate a tableau reasoner from this work into Swish. Current work-in-progress: http://www.ninebynine.org/Software/HaskellDL/.

Identified possible future work items include:

- integrated XML query and stylesheet processing for

scraping RDF data from arbitrary XML documents

- application to network device configuration and access control
- application to trust modelling (cf. http://www.ninebynine.org/iTrust/Intro.html)
- extension of RDF datatype-aware inference capabilities
- 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/)
- performance tuning.

I have a page of notes about my experience of learning Haskell at http://www.ninebynine.org/Software/Learning-Haskell-Notes.html.

Further information about my work is at http://www.ninebynine.org/ and http://www.ninebynine.net/.

8.3 Krasimir Angelov

Report by: Krasimir Angelov

The HToolkit (→ 4.5.4) and HSQL (→ 4.4.3) packages are still supported but there aren't too much new things here.

Currently I am involved in Visual Haskell (→ 5.5.3) project. The project aim is to provide plugin for Visual Studio which will allow to use Haskell inside the IDE. Since the Visual Studio integration API is COM based we use H/Direct to generate the FFI (→ 3.1) layer. In our work we found some bugs in H/Direct and now I am working to improve it.

8.4 Alain Crémieux

Report by: Alain Crémieux

I am working on a port to Haskell of a compiler for the Tiger toy language. The reference for the Tiger language is the book from Andrew Appel "Modern compiler implementation in ML". The corresponding code in ML is available on the Web, written by Yu Liao.

So the first step is a port of this code, but with the use of Haskell's tools Alex and Happy, up to the generation of machine code for a RISC processor. Then there will be 2 new outputs for the compiler, one directed at C– (a Tiger compiler generating C– is available, written in O'CAML by Paul Govereau), and the other towards LLVM, a virtual machine system (some examples of a Tiger compiler link to LLVM, written by Chris Lattner, are also available).

The roadmap is to add an intelligent editor for Tiger, in the style of Helium, and some kind of source level debugger, by reusing available components. All this should lead to a practical exemple of the complete implementation of a language, which is a domain in which Haskell is especially good. Any help & suggestions welcome, of course.

8.5 Inductive Inference

Report by: Lloyd Allison

Inductive Inference, i.e. the learning of general hypotheses from given data.

I am continuing to use Haskell to examine what are the products (e.g. Mixture-models (unsupervised classification, clustering), classification- (decision-) trees (supervised classification, expert systems), Bayesian/causal networks/models, etc.) of AIDMIIMLSI (= artificial-intelligence/ data-mining/ inductive-inference/ machine-learning/ statistical-inference/ etc.) 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 Haskell library or prelude for AIDMIIMLSI.

A JFP paper (see below) describes an early version of the software. 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. Recent case-studies include

- mixtures of time-series, and
- Bayesian networks.

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

Future plans

Try to find a good name for this kind of programming: 'function' is to 'functional programming' as 'statistical model' is to what?

I am currently developing time-series models further.

Further reading

- L. Allison. Models for machine learning and data mining in functional programming. J. Functional Programming, to appear 2004.
- Other reading is listed at the URL: http://www.csse.monash.edu.au/~lloyd/tildeFP/II/

8.6 Bioinformatics tools

Report by: Ketil Malde

I'm developing (what seems to become) a handful of tools for solving problems that arise in bioinformatics. I currently have a sequence clustering tool, xsact (currently in revision 1.4), which I believe is one of the more feature-rich tools of its kind. There is also a sequence assembly tool (xtract). In addition, there are various smaller tools that are or were useful to me, and that may or may not be, useful to others.

http://www.ii.uib.no/~ketil/bioinformatics