Haskell Communities and Activities Report

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

– sixth edition –

May 25, 2004

Arthur van Leeuwen (ed.) Krasimir Angelov Alistair Bayley Jérémy Bobbio
Björn Bringert Mark Carroll Manuel Chakravarty Olaf Chitil Andrew Cooke
Iavor Diatchki Peter Diviánszky Shae Erisson Levent Erkok Simon Foster
Andrew Frank Leif Frenzel Murray Gross Walter Guttmann Jurriaan Hage
Thomas Hallgren Keith Hanna Anders Höckersten Dean Herington Johan Jeuring
Oleg Kiselyov Graham Klyne Daan Leijen Andres Löh Ralf Lämmel Rita Loogen
Christoph Lüth Ian Lynagh Ketil Z. Malde Simon Marlow Serge Mechveliani
Brandon Moore Henry Nystrom André Pang Sven Panne Ross Paterson
Jens Petersen John Peterson Simon Peyton-Jones Jorge Sousa Pinto Bernie Pope
Alastair Reid Claus Reinke Frank Rosemeier David Roundy Chris Ryder
David Sabel Uwe Schmidt Axel Simon Ganesh Sittampalam Anthony Sloane
Dominic Steinitz Martin Sulzmann Wolfgang Thaller Peter Thiemann
Simon Thompson Phil Trinder Eelco Visser Malcolm Wallace Ashley Yakeley

Preface

After the initial scare that the May 2004 edition of the Haskell Communities & Activities Report might be without an editor, you see that any statements to the effect that the Report would no longer be a going concern were wildly exaggerated. It still muscles up to them eyes and vooms through the mind, showing how sizzling the community is. Somewhat clearer: you now behold the Sixth edition of the Haskell Communities and Activities report, a collection of contributions of Haskell enthousiasts worldwide, describing the things they are currently working on, and the plans they have for future work.

We are glad that this report has had even more contributions than the previous report, so that although a number of entries have been dropped, the total count has increased. This has also increased the total size of the report so that it is no longer easily read within half an hour. Whether that is a good thing or not is debatable — it does however show that many people are actively contributing. We would very much like even more contributions though! As mentioned before, contributing to the Haskell report is easy: a few words describing what the current status of your work is may well suffice. For examples, look at sections 4.7.2 and 6.1.3. Of course, **you** are best equipped to actually write those words, and we hope you will again, in time for the November edition of the Haskell Communities & Activities Report.

Once more, we remind you that these reports are not intended as a formal description of the state of Haskell. Quite the opposite in fact! They are intended as an informal look into the Haskell kitchen, a chance to let others smell all the interesting stuff you are cooking up. So please take out your diaries, and **mark the last weeks of October, as the contributions to the November edition are due by then**. Naturally, you can always send in reports to <hcar@haskell.org>, and we will then consult you in the weeks before the deadline to see if anything interesting has happened in the meantime.

This report shows how many-faceted the Haskell community really is. The applications of Haskell range from bio-informatics (section 8.3) through revision control (section 6.1.6) to Bayesian networks (section 6.1.11). A general trends emerge from the collection though: many people are using Haskell for practical programming, as can be seen from the increase not only in the number of Applications reported on, but also from the increased number of libraries dealing with user interfaces, foreign function interfaces and database interfaces. Of particular note is that there are now two efforts underway to incorporate Haskell programming in existing IDE's (sections 5.5.1, 5.5.2).

Furthermore, we are pleased to note that a fair number of contributions appear in this report for the first time. You may well be in for a few little surprises! Development of wxHaskell also seems to continue at an almost frightening pace, and some projects previously thought dead, such as HaskellDB, have been resurrected. Besides these exciting developments, it is good to see that the old standbys are still standing by. GHC is still going strong, as are Hugs and NHC.

All this implies a new surge in the uptake of Haskell, and seemingly from people new to the language, and quite excited by it. One testament to that is the interim editor of this report: just a bit more than a year ago I had not written any Haskell, and now I have had the great joy of compiling this report. Many thanks for all the encouraging words and signs of gratitude for doing this are in order, therefore: a big **thank you** to you all. The report would not have existed without you.

With that we leave you to read the report, and we hope you will come back in October to report to Andres as plentifully as you did for this report, and help him do the best job he can.

Once again, many thanks to all contributors, and have fun reading and programming!

Arthur van Leeuwen, University of Utrecht, NL

Contents

1 General **6**
1.1 `haskell.org` 6
1.2 `#haskell` 6
1.3 The Haskell HaWiki 7
1.4 Haskell related events 7

2 Implementations **8**
2.1 The Glasgow Haskell Compiler 8
2.2 Hugs 9
2.3 nhc98 9
2.4 hmake 9
2.5 Haskell-Clean Compiler 10
2.6 Domain-specific variations 10
2.6.1 Haskell on handheld devices 10
2.6.2 Helium 10
2.6.3 Educational Domain Specific Languages 10
2.6.4 Vital: Visual Interactive Programming 11
2.6.5 hOp 11

3 Language Extensions **12**
3.1 Foreign Function Interface 12
3.2 Non-sequential Programming 12
3.2.1 Concurrent Haskell 12
3.2.2 GpH – Glasgow Parallel Haskell 12
3.2.3 GdH – Glasgow Distributed Haskell 13
3.2.4 Eden 14
3.3 Type System/Program Analysis 15
3.3.1 Chameleon 15
3.3.2 Constraint Based Type Inferencing at Utrecht 16
3.4 Generic Programming 16
3.5 Template Haskell 18
3.6 Syntactic sugar 18
3.6.1 Recursive do notation 18
3.6.2 Arrow notation 18

4 Libraries **19**
4.1 Packaging and Distribution . . . 19
4.1.1 Library Infrastructure Project . . . 19
4.1.2 PreludeExts . . . 19
4.1.3 Haskel User Submitted Libraries . . . 19
4.2 General libraries . . . 20
4.2.1 System.Time: a redesigned Time library . . . 20
4.2.2 A redesigned IO library . . . 20
4.2.3 System.Process: a platform-independent API for external process control . . . 20
4.2.4 The Haskell Cryptographic Library . . . 21
4.2.5 Yampa . . . 21
4.2.6 The revamped monad transformer library . . . 21
4.2.7 HBase . . . 22
4.2.8 Pointless Haskell . . . 22
4.3 Parsing and transforming . . . 22
4.3.1 Parsec . . . 22
4.3.2 UPC – Utrecht Parser Combinators . . . 22
4.3.3 Strafunski . . . 23
4.3.4 Medina – Metrics for Haskell . . . 23
4.3.5 Template Greencard . . . 24
4.4 Data handling . . . 24
4.4.1 DData . . . 24
4.4.2 HSQL . . . 24
4.4.3 Takusen . . . 24
4.4.4 HaskellDB . . . 25
4.5 User interfaces . . . 25
4.5.1 The Common GUI API effort . . . 25
4.5.2 wxHaskell . . . 25
4.5.3 HToolkit . . . 26
4.5.4 gtk2hs - A binding to the Gtk GUI library version 2.0 - 2.4. . . . 26
4.5.5 HTk . . . 26
4.5.6 HSX11 . . . 26
4.5.7 Fudgets . . . 27
4.6 Graphics . . . 27
4.6.1 HOpenGL – A Haskell Binding for OpenGL and GLUT . . . 27
4.6.2 FunWorlds – Functional Programming and Virtual Worlds . . . 27
4.6.3 PanTHeon . . . 28
4.6.4 Pancito . . . 28
4.7 Web and XML programming . . . 28
4.7.1 Halipeto . . . 28
4.7.2 HaXml . . . 29
4.7.3 Haskell XML Toolbox . . . 29
4.7.4 WASH/CGI – Web Authoring System for Haskell . . . 29
4.7.5 HAIFA . . . 30
4.7.6 Haskell XML-RPC . . . 30

5 Tools **31**
5.1 Foreign Function Interfacing . . . 31
5.1.1 GreenCard . . . 31
5.1.2 C–>Haskell . . . 31
5.1.3 JVM Bridge . . . 31
5.1.4 PHI – Python Haskell Interface . . . 31
5.1.5 HOC: A Haskell to Objective-C binding . . . 32
5.2 Scanning, Parsing, Analysis . . . 32
5.2.1 Alex version 2 . . . 32
5.2.2 Happy . . . 32
5.2.3 HaLex . . . 32
5.2.4 LRC . . . 33
5.2.5 Sdf2Haskell . . . 33
5.2.6 The Utrecht attribute grammar system UAG . . . 33
5.2.7 DrHylo . . . 33
5.3 Transformations . . . 33
5.3.1 The Programatica Project . . . 33
5.3.2 Ultra . . . 34
5.3.3 Hare – The Haskell Refactorer . . . 34
5.3.4 VooDooM . . . 34
5.4 Testing and Debugging . . . 35
5.4.1 Tracing and Debugging . . . 35
5.4.2 Hat . . . 35
5.4.3 buddha . . . 35
5.4.4 HUnit . . . 36
5.5 Development . . . 36
5.5.1 Visual Studio support for Haskell . . . 36
5.5.2 Haskell support for the Eclipse IDE . . . 36
5.5.3 Haddock . . . 36

6 Applications **37**
6.1 Non-commercial applications . . . 37
6.1.1 HScheme . . . 37
6.1.2 Curryspondence . . . 37
6.1.3 lambdabot . . . 37
6.1.4 HWS-WP . . . 37
6.1.5 Hircules, an irc client . . . 38
6.1.6 Darcs . . . 38
6.1.7 Yarrow . . . 38
6.1.8 HasLaTeX . . . 38
6.1.9 DoCon, the Algebraic Domain Constructor . . . 38
6.1.10 lhs2TEX . . . 39
6.1.11 NetEdit . . . 39
6.2 Commercial users . . . 39
6.2.1 Reid Consulting Ltd . . . 39
6.2.2 Aetion Technologies LLC . . . 39
6.3 Haskell in Education . . . 40
6.3.1 Haskell in Education at Universidade de Minho . . . 40
6.3.2 Beseme Project . . . 40

7 Groups **42**
7.1 Research Groups . . . 42
7.1.1 Artificial Intelligence and Software Technology at JWG-University Frankfurt . . . 42
7.1.2 Formal Methods at Bremen University . . . 43
7.1.3 Functional Programming at Brooklyn College, City University of New York . . . 43
7.1.4 Functional Programming at Macquarie University . . . 43
7.1.5 Functional Programming at the University of Kent . . . 44
7.1.6 Parallel and Distributed Functional Languages Research Group at Heriot-Watt University . . . 44
7.1.7 Programming Languages & Systems at UNSW . . . 45
7.1.8 Institute for Geoinformation at TU Vienna . . . 45
7.1.9 Logic and Formal Methods group at the Informatics Department of the University of Minho, Braga, Portugal . . . 46
7.1.10 Functional Programming at Utrecht University . . . 46
7.2 Other groups . . . 47
7.2.1 Debian Users . . . 47
7.2.2 Fedora Haskell packages . . . 48

8 Individuals **49**
8.1 Oleg's Mini tutorials and assorted small projects . . . 49
8.2 Graham Klyne . . . 49
8.3 Bioinformatics tools . . . 50

Chapter 1

General

1.1 haskell.org

Report by: *John Peterson*

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

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.

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 stuff.

1.4 Haskell related events

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

SBLP 2004 O 8o Simpósio Brasileiro de Linguagens de Programação sera... my apologies. The 8th Brazilian Symposium on Programming Languages will be held at the Universidade Federal Fluminense in the city of Niterói, Brazil, between the 26th and the 28th of May. See `http://sblp2004.ic.uff.br`.

ICFP Programming contest Do you want to have your favorite language declared 'A fine tool for discerning hackers', and gather eternal fame? Participate in the Seventh ICFP Programming Contest, starting June 4th 2004 and lasting for 72 hours. See `http://www.cis.upenn.edu/proj/plclub/contest/`.

EuroHaskell Less talks, more code! On june 10th through 12th, European Haskell developers are welcome in Gothenburg to meet and code. See also `http://www.haskell.org/hawiki/EuroHaskell`.

The Succ Zero IOHCC The Succ Zero International Obfuscated Haskell Code Contest is coming soon to a website near you!
`http://www.ScannedInAvian.org/iohcc`

AFP 2004 To learn advanced functional programming techniques from those that have developed them, what is a better place than the Summer School on Advanced Functional Programming? The 2004 edition will be held in the beautiful Estonian city of Tartu, from the 14th to the 21st of August. See `http://www.cs.ut.ee/afp04/`.

IFL'04 The 16th International Workshop on Implementation and Application of Functional Languages provides you with an excellent forum to present your work. It will be held in Lübeck, Germany, on September 8th through 10th. See `http://www.isp.uni-luebeck.de/ifl04/Deadlines/index.htm`.

ICFP04 The big one, the International Conference on Functional Programming, this year will be held in Snowbird, Utah, USA, from the 19th to the 22nd of September. See `http://www.cs.indiana.edu/icfp04/`.

Haskell Workshop 2004 Next to the ICFP, on the 22nd of September, the annual Haskell Workshop awaits your input, also in Snowbird, Utah, USA. See `http://www.cs.nott.ac.uk/~nhn/HW2004/`.

TFP'04 On an undisclosed date somewhere this year, Munich will provide a venue for the Symposium on Trends in Functional Programming. For more information, see `http://www.cee.hw.ac.uk/~dsg/sfp/`.

Chapter 2

Implementations

2.1 The Glasgow Haskell Compiler

Report by: *Simon Peyton-Jones*

GHC status (April 2004) We released GHC 6.2 in December 2003, with a bug-fix release of 6.2.1 in April.

Here are some development highlights from the last few months. None of them are in the released compiler yet – indeed not all of the developments described in the November 2003 Report are in 6.2 either. We expect to release 6.4, with all this new stuff, in May or June.

- In January and February we did a complete rewrite of GHC's back end. The hoary old data types `AbstractC` and `Stix` have gone, and in their place is a new, much smaller data type `Cmm`. This data type is inspired directly by the `C--` language, and indeed, we can now generate `C--` as well as `C` and native code.

  Not only is the result far cleaner and easier to maintain, but

  1. it's at least 1500 lines shorter
  2. everything works via the native code generator route (previously profiling and ticky-ticky counting did not)

  Final step: replace the old "`.hc`" files in the runtime system with shiny new `C--` files, and persuade GHC to parse them into `Cmm`. Then we can (optionally) abandon the via-C route altogether if we want.

  Many thanks to Don Stewart, who has helped a lot with this stuff.

- Further developments on the scrap-your-boilerplate front are fully implemented. A new paper is now available at `http://research.microsoft.com/~simonpj/papers/hmap/gmap2.htm`

- A slew of Template Haskell improvements, notably the introduction of an abstract type `Name`, as envisaged in the (still rather scrappy) design note `http://research.microsoft.com/~simonpj/tmp/notes2.ps`.

- On Windows, GHC can now readily be built from source using MSYS, as well as Cygwin. MSYS is a cut-down version of Cygwin; the build goes significantly faster.

- A few improvements to the way in which overlapping instances are handled. In particular, overlap is now checked lazily. So it's OK to have:

  ```
  instance C Int b where ...
  instance C a Int where ...
  ```

  Then if you need `(C Int Bool)`, the first applies unambiguously, and there is no problem. Only if you need `(C Int Int)`, which matches both, is an overlap reported.

Future plans:

- Simon M. will do more work on the GHC Visual Studio plug-in.

- Geoffrey Washburn is coming to Cambridge for an internship. We plan to implement so-called "first-class phantom types", also called "guarded recursive data types" (Xi), "inductive types" (Coq), and "equality-qualified types" (Sheard).

- We still are actively working with Isaac Jones on plans for library packaing, and will adapt GHC to work with the new scheme.

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 helping to 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*

Hugs is a very portable, easily installed Haskell-98 compliant interpreter that supports a wide range of type-system and runtime-system extensions including typed record extensions, implicit parameters, the foreign function interface extension and the hierarchical module namespace extension.

Current State

The most recent release of Hugs was in November 2003, marking a shift to the Haskell hierarchical libraries, though many old libraries are implemented by compatibility stubs. Some day these old interfaces will disappear; users are encouraged to migrate to the new interfaces, which are more powerful and offer greater compatibility with other Haskell implementations.

The development version incorporates support for Unicode, thanks to Dimitry Golubovsky <dimitry@golubovsky.org>. There is also a steady trickle of fixes and minor enhancements.

Future plans:

The manpower available for Hugs development and maintenance remains very limited. Contributions from volunteers are welcome. We would particularly like to hear from people prepared to build, test and debug on Windows. (A full build requires one of the free Unix-like environments for Windows.)

We would like to do the next release in the summer, in time for classes in the northern hemisphere, but we've not managed that in the last few years. (See the request for volunteers above.)

The next release will probably 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. We will also cooperate with the Library Infrastructure Project (section 4.1.1), and will use it if it is ready in time. Library authors who would like to participate should make their libraries work with Hugs and contact us.

2.3 nhc98

Report by: *Malcolm Wallace*
Status: *stable, maintained*

nhc98, a small, easy to install, standards-compliant compiler for Haskell 98, is in stable maintenance-only mode. The public release remains at version 1.16 for the moment. Maintenance and bugfixes continue to the CVS tree at haskell.org. When sufficient serious fixes have accumulated, a new public release will be forthcoming. No innovative new features are currently planned, unless you, dear reader, would like to volunteer!

Further reading:

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

2.4 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`.

Future plans:

The hmake source code is being used as a basis for parts of the new library infrastructure project. It has also recently contributed code to 'cpphs', a minimal replacement in Haskell for the C-preprocessor. The latter will hopefully develop further into a useful tool in its own right.

Further reading:

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

2.5 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.

2.6 Domain-specific variations

2.6.1 Haskell on handheld devices

Report by: *Anthony Sloane*
Status: *unreleased*

Work on our port of nhc982.3 to Palm OS is continuing but, unfortunately, is not ready for public release at this stage. In our revised schedule we plan to have something released by the end of this year.

2.6.2 Helium

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

The purpose of the Helium project is to construct a lightweight compiler for a subset of Haskell that is especially directed to beginning programmers (see *"Helium, for learning Haskell"*, Bastiaan Heeren, Daan Leijen, Arjan van IJzendoorn, Haskell Workshop 2003). We try to give useful feedback for often occurring mistakes. To reach this goal, Helium uses a sophisticated type checker described in section 3.3.2 (see also *"Scripting the type inference process"*, Bastiaan Heeren, Jurriaan Hage and S. Doaitse Swierstra, ICFP 2003).

Helium now has a simple graphical user interface that provides online help. We plan to extend this interface to a full fledged learning environment for Haskell. The complete type checker and code generator has been constructed with the attribute grammar (AG) system developed at Utrecht University (section 7.1.10) One of the aspects of the compiler is that it also logs errors, so we can track the kind of problems students are having, and improve the error messages and hints.

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 June 2004.

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 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 or graphically (as linked data structures), and can be edited using conventional Copy/Paste mouse gestures. This gives end users a convenient, 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.

A recent development allows Vital documents to include hyperlinked comments, allowing a user to navigate freely between documents without needing to be concerned about loading them explicitly.

A collection of interactive tutorial documents (including a couple illustrating approaches to Exact Real Arithmetic) have recently been added.

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

Further reading:

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

2.6.5 hOp

Report by: *Jérémy Bobbio*
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 release of hOp consists 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 makes most functions from the base hierarchical library available (all but the System modules), including support for threads, communication primitives, and the foreign function interface (FFI).

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.

We are currently working towards integrating GHCi to the system, to be used as a shell. We plan to add the necessary features for the system to be able to compile itself (bootstrap), including some kind of data storage facility, an assembler and a linker (all of course implemented in Haskell).

Further reading:

Further information and source code are available here: `http://www.macs.hw.ac.uk/~sebc/hOp/`

Chapter 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 has finally been released. 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/`

As the editor of the report, it is my pleasure to thank the many people who have contributed to the FFI standard by proposing designs, reading and correcting the report, implementing the various revisions of the standard, and by testing it in their software. As a result of the broad involvement, the FFI Addendum is a community product, just like Haskell itself.

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 Concurrent Haskell

Report by: *Wolfgang Thaller*

The threaded RTS for GHC, including the new "bound threads" feature, has become part of the standard distribution of GHC, starting with 6.2.1; the feature is activated by passing a command-line flag (-threaded) to GHC when linking.

3.2.2 GpH – Glasgow Parallel Haskell

Report by: *Phil Trinder*

The Team

Phil Trinder, Kevin Hammond, Hans-Wolfgang Loidl, Abyd Al Zain, Jost Berthold, Xiao Yan Deng, Murray Gross, Steffen Priebe, Andre Rauber du Bois, Leonid Timochouk, Yang Yang.

Status

A complete, GHC-based implementation of the parallel Haskell extension GpH (`http://www.macs.hw.ac.uk/~dsg/gph/#GPH`) and of evaluation strategies (`http://www.macs.hw.ac.uk/~dsg/gph/papers/html/Strategies/strategies.html`) is available. Extensions of the runtime-system and language to improve performance and support new platforms are under development.

System Evaluation and Enhancement

The first 3 items are linked by a British Council/DAAD project (`http://www.macs.hw.ac.uk/~dsg/projects/GpHGRID.html`).

- We have ported GUM to computational GRIDs, replacing the current PVM communications layer with MPICH-G2. Measurements show that standard GUM gives good performance on a single cluster and on multiple clusters with low-latency interconnect. However standard GUM gives poor performance on multiple clusters connected with high-latency interconnect and we are developing GRIDGum - an implementation with specific adaption techniques for this shared, heterogeneous and hierarchical architecture.

- We are designing a generic parallel runtime environment encompassing both the Eden (section 3.2.4) and GpH runtime environments
- We are investigating cost models for distributed computations designing an integrated runtime environment for both Eden and GpH.
- 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 (`http://www.macs.hw.ac.uk/~trinder/ParDistr/` and Phillips Universität Marburg (`http://www.mathematik.uni-marburg.de/~loogen/Lehre/ws02/pfp/vor02WSpfp.shtml`).

GpH Applications

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

Implementations

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

- The stable branch (GUM-4.06, based on GHC-4.06) is available for RedHat-based Linux machines: binary snapshot at `ftp://ftp.macs.hw.ac.uk/pub/gph/gum-4.06-snap-i386-unknown-linux.tar` (see installation instructions in `README.GUM`). 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 currently being tested on a Beowulf cluster. The unstable branch is available from the GHC CVS repository via tag `gum-5-02-3`.

Our main hardware platform are Intel-based Beowulf clusters. Work on ports to other architectures is also moving on (and available on request from <gph@macs.hw.ac.uk>):

- A port to a Sun-Solaris shared-memory machine exists but currently suffers from performance problems.
- A port to a Mosix cluster has been built in the Metis (section 7.1.3) project at Brooklyn College, with a first version available on request from Murray Gross (<mgross@dorsai.org>).

Further reading:

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

3.2.3 GdH – Glasgow Distributed Haskell

Report by: *Jan Henry Nyström*
The Team: *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 (Glasgow parallel Haskell; section 3.2.2).

GdH Applications and Evaluation

- An EPSRC project *High Level Techniques for Distributed Telecommunications Software* (`http://www.macs.hw.ac.uk/~dsg/telecoms/`, GR/R88137) is now underway and is entering its first GdH phase. The project evaluates GdH and Erlang in a telecommunications context, the work is 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:

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

3.2.4 Eden

Report by: *Rita Loogen*

Description.

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

in Madrid: Ricardo Peña, Yolanda Ortega-Mallén, Alberto de la Encina, 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), synchronization, and process handling.

Eden's main constructs are process abstractions and process instantiations. The function `process :: (a -> b) -> Process a b` embeds a function of type `(a -> b)` into a *process abstraction* of type `Process a b` which, when instantiated, will be executed in parallel. *Process instantiation* is expressed by the predefined infix operator `( # ) :: Process a b -> a -> b`. Higher-level coordination is achieved by defining *skeletons*, ranging from a simple parallel map to sophisticated replicated-worker schemes. They have been used to 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 via the ghc CVS repository with tag eden-5-02-3. We are eager to catch up to the current ghc version.

Recent Publications

survey:

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

runtime-system level optimisations:

Jost Berthold: *Dynamic Chunking in Eden*, Proceedings of Implementation of Functional Languages, IFL 2003, Edinburgh (UK), September 2003, Springer LNCS, to appear.

generalised runtime system:

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

Eden-Maple interface:

Rafael Martínez and Ricardo Peña: *Building an Interface Between Eden and Maple: A way of Parallelizing Computer Algebra Algorithms*, Proceedings of Implementation of Functional Languages, IFL 2003, Edinburgh (UK), September 2003, Springer LNCS, to appear.

semantics:

1. Mercedes Hidalgo-Herrero and Yolanda Ortega-Mallén: *Continuation Semantics for Parallel Haskell Dialects*, APLAS'03 — the First Asian Symposium on Programming Languages and Systems, Beijing, China, November 27–29, Springer LNCS.
2. M. Hidalgo-Herrero: *Formal Semantics for a parallel functional language*, Ph.D. Thesis, Universidad Complutense de Madrid 2004 (in spanish).

Current Activities

- Yolanda and Mercedes have developed a denotational semantics for Eden which is based on a continuation-based model for process creation and single-value communication. Mercedes has finished her Ph.D. thesis on formal semantics for a parallel functional language.
- Rafael and Ricardo do experiments with computation-intensive computer-algebra algorithms on the recently developed Eden-Maple interface.
- Jost is working on a new, more general implementation of parallel Haskell dialects in a shared runtime system. Starting point is the support for Eden in GHC 6.x, but the overall target is a generic parallel platform that can

support multiple high-level languages and that offers implicit control of key runtime aspects such as thread management, synchronization and communication.

- The use of Template Haskell to improve or simplify the compilation of Eden programs will be investigated by the Marburg group. In particular, Steffen's work on the polytypic skeleton library for Eden benefits from the new meta-programming facilities.
- Pablo has extended the Eden runtime environment to output trace information which can be visualized by a newly developed trace viewer.

Further reading:

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

3.3 Type System/Program Analysis

3.3.1 Chameleon

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

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.

In our most recent work we focus on the following topics:

Type annotations in Haskell: We consider type inference in Haskell in the presence of type annotations, i.e. user-provided type declarations. Existing implementations such as HUGS and GHC have some clear limitations. Some programs are rejected despite being well-typed. We propose two novel inference schemes which significantly improve over previous formulations. One is based on logic formulae. We can state some sufficient conditions under which we achieve principal types. In general, inference via logic formulae is not feasible. Therefore, we propose a slightly weaker formulation in terms of Constraint Handling Rules (CHRs). The CHR-based inference system has been fully implemented as part of the Chameleon system (experimental version of Haskell).

Improving Type Error Diagnosis: We present a number of methods for providing improved type error reports in the Haskell and Chameleon programming languages. We build upon previous work (see Haskell'03 paper) where we first introduced the idea of discovering type errors by translating the typing problem into a constraint problem and looking for minimal unsatisfiable subsets of constraints. This allowed us to find precise sets of program locations which are in conflict with each other. Here we extend this approach by extracting additional useful information from these minimal unsatisfiable sets. This allows us to report errors as conflicts amongst a number of possible, candidate types. The advantage of our approach is that it offers implementors the flexibility to employ heuristics to select where, amongst all the locations involved, an error should be reported. In addition, we present methods for providing improved subsumption and ambiguity error reporting.

Functional dependencies (FDs) are a popular and useful extension to Haskell style type classes. We gave a reformulation of functional dependencies in terms of CHRs which has the following merits:

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

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 Haskell error messages in mind we have devised a constraint based type inference method which is currently being used in the Helium compiler (section 2.6.2) developed at Universiteit Utrecht.

The main characteristics of the inferencer are the following.

- Our philopsophy is that no single type inferencer works best for everybody all the time. Hence, we want a tunable type inferencer adaptable to the programmer's needs without the need for him to delve into the compiler.

- We generate precise position information and preserve type synonyms in error messages.

- The programmer can choose the type inference strategy of his liking (M and W and other greedy variants, and the unbiased type graph based implementations have been implemented).

- The type graph implementation uses quite a number of heuristics to decide what is the most likely source of the error.

- A logging facility is available in Helium which has given us a large amount of correct and erroneous Haskell programs which can be used to improve our type inferencer. In the future these programs can also be used for benchmarking optimizations and many other purposes. The programs have been anonymized, but the relation between programs by the same programmer has been kept intact. Various questions can then be answered: Do our hints help? Are they used? It is easy to come up with many interesting questions. Currently we have information from two instances of a first year programming course.

- A major innovation is the ability for a programmer to develop his domain specific type rules for a combinator library he might be writing. In addition, he may specify that his experiences are that certain functions are often mixed up. As a result, a compiler may give the hint that (++) should be used instead of (:), because (++) happens to fit in the context.

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

  An article on this facility can be found in the ICFP '03 proceedings.

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 Cork, Ireland.

Since the report of Nov 2003

- We have separated the general aspects of the solver from the Helium type inferencer so that a lot of the functionality can be used in other projects too. A notable example, on which a student is currently working, is to perform constraint based strictness analysis for Haskell.

- We have been working on type inference directives for type classes and have added type classes information to the already existing type inference directives and heuristics. We expect to report on this soon.

- Second, 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.

- On the technical level, we have made some progress with making the underlying machinery more elegant. Some of these changes are instigated by a need for integrating the various 'phases' of type inferencing such as the unification phase, the reduction of class constraints, the checking for 'declared type too general' and so on.

Further reading:

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

3.4 Generic Programming

Report by: *Johan Jeuring*

Software development often consists of designing a (set of mutually recursive) datatype(s), to which functionality is added. Some functionality is datatype specific, other functionality is defined on almost all datatypes, and only depends on the type structure of the datatype.

Examples of generic (or polytypic) functionality defined on almost all datatypes are the functions that can be derived in Haskell using the deriving construct, storing a value in a database, editing a value, comparing two values for equality, pretty-printing a value, etc. Another kind of generic function is a function that traverses its argument, and only performs an action at a small part of its argument. A function that works on many datatypes is called a generic function.

There are at least two approaches to generic programming: use a preprocessor to generate instances of generic functions on some given datatypes, or extend a programming language with the possibility to define generic functions.

Preprocessors

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

Languages

Light-weight generic programming: Generic functions for data type traversals can (almost) be written in Haskell itself, as shown by Ralf 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 submitted paper "Scrap more boilerplate: reflection, zips, and generalised casts" available from `http://www.cs.vu.nl/boilerplate/`. This papers 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", Ralf Hinze shows how to write generic programs in Haskell98, without any fancy extensions. See `http://www.informatik.uni-bonn.de/~ralf/`.

Generic Haskell: 'Dependency-style' Generic Haskell introduces a new type system for Generic Haskell that at the same time simplifies the syntax and provides greater expressive power, see the ICFP paper by Andres Löh, Dave Clarke and Johan Jeuring for a description. A type-checker has been implemented for dependency-style Generic Haskell. Andres Löh will defend his PhD thesis on this topic this summer, electronic copies will be available when the thesis is ready.

Generic Haskell is used in 'UUXML: A Type-Preserving XML Schema - Haskell Data Binding' by Frank Atanassow, Dave Clarke and Johan Jeuring (to appear in 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 (to appear in MPC'04).

A new generation of PolyP has seen the light of day accompanied by a paper "Generic programming in Haskell" by Norell and Jansson (in submission - presented at IFL'03). The new approach embeds polytypic functions in Haskell using a type classes. This means that PolyLib, the library of polytypic functions, is now available as a Haskell library. Thus the separate PolyP compiler is not strictly needed anymore. (The compiler provides a more convenient syntax for definition of new polytypic functions and it automatically derives instances for regular datatypes.) Furthermore, Ulf Norell and Patrik Jansson at Chalmers have been working on "Prototyping Generic Programming in Template Haskell" (paper accepted for MPC 2004).

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 (to appear in MPC'04). Martijn de Vries (University of Groningen) has written an MSc thesis on how to apply similar techniques in Generic Haskell (email the report author for a copy).

The Datatype-Generic Programming project at Oxford and Nottingham (`http://www.comlab.ox.ac.uk/oucl/research/areas/ap/dgp`) started in August 2003, with aims, amongst others, to develop a methodology for constructing generic programs. Jeremy Gibbons and Bruno Oliviera are studying different approaches for generic programming based on Haskell; Roland Backhouse and Fermin Reig are looking at termination properties of generic functions.

Current Hot Topics: Generic Haskell: incorporating views on data types in the language. 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: Efficient generic traversal based on type-information for premature termination (see the Strafunski project). Exploring the differences in expressive power between the lightweight approaches and the language extension(s).

Further reading:

`http://repetae.net/john/computer/haskell/DrIFT/`
`http://www.cs.chalmers.se/~patrikj/poly/`
`http://www.generic-haskell.org/`
`http://www.cs.vu.nl/Strafunski/`
`http://www.cs.vu.nl/boilerplate/`

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

3.5 Template Haskell

Report by: *Ian Lynagh*

Template Haskell has been growing steadily over the past 6 months, with support for various GHC extensions being added on demand. If you find yourself needing something that isn't yet implemented, please let us know at <template-haskell@haskell.org>. The good work of Simon Peyton Jones has seen a number of new features, including:

- Prefixes `'` and `''` to get (non-monadic) Names of values and types, e.g. `'map :: Name` and `''Int :: Name`
- Additional reification information from a normal function `reify` of type `Name -> Q Info`
- Error reporting and recovery functions

```
report :: Bool {- True <=> fatal -}
       -> String
       -> Q ()
recover :: Q a -> Q a -> Q a
```

plus of course the odd bug-fix!

The naming seems to have stabilized now, so if you haven't experimented with Template Haskell already then perhaps this would be a good time to do so! And we're always interested to hear what people are using it for.

3.6 Syntactic sugar

3.6.1 Recursive do notation

Report by: *Levent Erkok (nov 2003)*
Status: *stable*

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

Further reading:

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

3.6.2 Arrow notation

Report by: *Ross Paterson (nov 2003)*
Status: *stable*

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

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

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

Further reading:

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

Chapter 4

Libraries

4.1 Packaging and Distribution

4.1.1 Library Infrastructure Project

Report by: *Isaac Jones*

Background:

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

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

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

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

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

Status:

After several prototypes, Isaac Jones visited GHC Headquarters in Cambridge and met with Simon Peyton Jones, Simon Marlow, Malcolm Wallace, and Ross Patterson to come to a consensus on implementation details. It was pretty fun :). A new proposal is nearly ready for release, and Isaac and Simon Marlow have done some hacking on it already. We expect a beta release in the near future.

Further reading:

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

4.1.2 PreludeExts

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.
`http://www.haskell.org/hawiki/PreludeExts`

4.1.3 Haskel User Submitted Libraries

Report by: *Shae Erisson*

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

This sourceforge project is open for any haskell developers who wish to contribute: `http://sf.net/projects/haskell-libs/` We also have an experimental darcs repository on `http://www.ScannedInAvian.org/repos/hlibs/`

Some current projects that are available there:

lambdabot an IRC bot (section 6.1.3)

hws-wp the hws webserver extended with plugins (section 6.1.4)

brainfuck a brainfuck interpreter

pdflib a binding to PDFLib Lite

takusen win32 Oracle binding (section 4.4.3)

4.2 General libraries

4.2.1 System.Time: a redesigned Time library

Report by: *Simon Marlow*

Almost a year ago there was much discussion about a replacement for the current `Time` library, because of certain problems with the existing library:

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

The latest proposal was posted to the libraries@haskell.org mailing list 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 is stalled. 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.

4.2.2 A redesigned IO library

Report by: *Simon Marlow*

Contributors: *Ben Rudiak-Gould, Simon Marlow and others*

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

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

Simon M. has been hacking a little on a prototype, but doesn't have anything significant working yet.

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

Report by: *Simon Marlow*

A proposal for a System.Process library was posted to the <libraries@haskell.org> list about a year ago:
`http://haskell.org/pipermail/libraries/2003-May/000958.html`

HTML documentation is here:
`http://www.haskell.org/~simonmar/System.Process.html`

Further progress has been made in that there is a partial implementation of this library, currently restricted to GHC running on Unix platforms. The code is here: `http://www.haskell.org/~simonmar/process/Process.hs`

Once a Windows implementation is available, this library will be imported into the hierarchical libraries.

4.2.4 The Haskell Cryptographic Library

Report by: *Dominic Steinitz*

The current release is 1.1.2. New, since the last report, are the additions of MD5 and modules to support ASN.1 and PKCS#8. In addition, the library now runs under Hugs (courtesy of Ross Paterson) as well as GHC.

The library collects together existing Haskell cryptographic functions and augments them so that they: a) have common type signatures and b) can be used with the standard mode and padding algorithms (in the case of block mode ciphers). The library now supports: DES, Blowfish, Cipher Block Chaining (CBC) mode, PKCS5 and nulls padding, MD5, SHA-1, RSA, OAEP, ASN.1 and PKCS#8. The RSA and OAEP modules are based on the work done by David Sankel.

The library follows the hierarchical standards and has Haddock style documentation. There are demo / test programs using published test vectors and instructions on how to use RSA in Haskell and inter-work with openssl. 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) and AES. Furthermore, you cannot use RSA to encrypt (easily) as there is currently no support for X.509. Nor is there any support for signing.

Further reading:

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

4.2.5 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`

4.2.6 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. It is available from the Haskell CVS (fptools/libraries/monads). It is mostly documented with haddock (section 5.5.3).

Besides reorganizing the transformers, the main changes include a new experimental transformer for non-determinism, renaming of some functions, and some new functionality here and there. There is also experimental support for resumptions, and continuations, but their interaction with the other transformers is not quite clear at the moment, and the API has not been finalized. Please try to use the library and comment on how useful it is to you!

Further reading:

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

4.2.7 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 (section 6.1.1) and my other Haskell projects, and sometimes by whatever interests occur to me. Right now it includes:

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

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

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 (section 5.1.3) have been cleared up.

Further reading:

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

4.2.8 Pointless Haskell

Report by: *Jorge Sousa Pinta*

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. 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 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).

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

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 UPC – Utrecht Parser Combinators

Report by: *Doaitse Swierstra*

(Doaitse Swierstra, Arthur Baars, Rui Guerra)

The current version of the parser combinators constructs an online result, in the sense that parts of the result can be accessed even when parsing has not yet finished. This is especially useful when parsing and processing large files of similar information. Furthermore error messages are displayed while parsing (using unsafePerformIO). The underlying mechanism for achieving this is relatively costly, although parsing speed is not much slower than that of parsers generated off line using Frown or Happy (section 5.2.2). We plan to construct a companion module (based on an earlier approach) that contains a more strict result, and which we expect to be running even faster. Furthermore the module structure may be changed by making it possible for the user of the library to tune the internals of the machine even more using classes. In order to make the everyday use of the combinators not suffer from these changes we have separated the interface and the extensions from the basic implementation, so future changes can relatively easily be made.

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

Further reading:

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

4.3.3 Strafunski

Report by: *Ralf Lämmel*

Status: *active, maintained, new release in April 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 (section 3.4);
- the library ATermLib for data exchange;
- the tool Sdf2Haskell for external parser integration.

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

Strafunski has now moved to sourceforge.net. A new version (4.0) of the Strafunski's StrategyLib was released. In due the course, the distribution was simplified, and optional support for a model based on "Scrap your boilerplate" has been added.

There is a new white paper "Programmable rewriting strategies in Haskell" (`http://homepages.cwi.nl/~ralf/wrs04/`), which puts Strafunski into a broader perspective, which lists several applications of Strafunski, and which answers some frequently asked questions. Strafunski is also covered in an upcoming tutorial by Joost Visser and João Saraiva "Strategic Programming Across Programming Paradigms" at SBLP 2004, the 8th Brazilian Symposium on Programming Languages.

Further reading:

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

4.3.4 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 I am in the process of writing up this work for my PhD thesis. Once this task is completed I will prepare a release of the library.

4.3.5 Template Greencard

Report by: *Alastair Reid*

Status: *Experimental/unstable*

Last release: *0.1 (15 Sept 2003)*
Hierarchical libraries: *No*
Portability: *GHC only (requires Template Haskell), Unix*

Template Greencard is an experimental reimplementation of Greencard (section 5.1.1) using Template Haskell. It is now considered dead, however.

Its advantages are:

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

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

Further reading:

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

4.4 Data handling

4.4.1 DData

Report by: *Daan Leijen*
Status: *actively maintained, 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 HSQL

Report by: *Krasimir Angelov*

The HSQL is a simple library for database access from Haskell. The HSQL library was completely rewritten in the last few months. There are four major steps forward:

- The library now provides an abstract application programming interface which allows us to write database independent applications.
- In addition to ODBC, MySQL and PostgreSQL backends now there is also a backend for SQLite. The SQLite is a small and fast embedded database which is suitable for cases when it is not required to have multiple clients for a single database.
- Thanks to Björn Bringert the library was successfully ported to Hugs.
- Thanks to Victor Blomqvist and Conny Andersson in the last HSQL release there is RPM distribution for RedHat and a binary installer for Windows.

URL: `http://htoolkit.sourceforge.net/`

4.4.3 Takusen

Report by: *Alistair Bayley, Oleg Kiselyov*

Takusen is a library for accessing DBMS's. It is a low-level library like HSQL, in the sense that it is used to issue SQL statements. It currently only supports one 'real' DBMS (Oracle), but is designed to easily support other DBMS's. In particular, there is a stub module which is used for testing in the absence of an Oracle installation, and which can be easily extended for other DBMS's. 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.

Takusen is under active development. Recently we have modified the library to use monadic iteratee functions, which allow the user to do IO while processing query results. Currently we are assessing the performance of the library with large result sets.

Immediate plans are to fix deficiencies in the current implementation (this includes performance tuning, bind variables, multiple connections, and os-authenticated logon), and in the more distant future to add support for other DBMS's.

Source code can be found in the Haskell-libs project at SourceForge: `http://cvs.sf.net/viewcvs.py/haskell-libs/libs/takusen/`

4.4.4 HaskellDB

Report by: *Anders Höckersten*

Status: *active development*

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 or wxHaskell.

HaskellDB was originally developed by Daan Leijen. Development was restarted as part of a student project at Chalmers University of Technology. This project is nearly over, but several of us have expressed interest in continuing development in our spare time. There is also a community of developers using HaskellDB that constantly send us patches and improvements.

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

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
- Dynamic loading of drivers

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 built 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 85% of the wxWindows functionality – 2875 methods in 513 classes with 1347 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 Communities & Activities Report, most work has been directed to improved stability and a better build system. There is also better integration with other packages: HaskellDB works with the wxHaskell ODBC binding and HOpenGL 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 this summer.

Further reading:

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

4.5.3 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 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. The package development was suspended as the Linux version was dependent on GTK-2.4. The GTK-2.4 release was delayed for a long time but now it is completed and I am ready to continue the development of HToolkit.

URL: `http://htoolkit.sourceforge.net/`

4.5.4 gtk2hs - A binding to the Gtk GUI library version 2.0 - 2.4.

Report by: *Axel Simon*

This project provides a high-quality binding to the Gtk GUI library together with some Gnome extensions (at the moment Glade and GConf). It predates wxHaskell (which can use Gtk as a back-end) but might be of interest to people focussing on the Gnome desktop or who are planning to write large applications where automatic memory management is a requirement (wxWidgets does not provide proper support for garbage-collected languages). The binding is not automatically generated which has the advantage that we notice changes in the Gtk API when we build the library (as supposed to building an application). The library is and will be maintained, in particular we are planning to add more Gnome widgets and add a Yaho/Ports like layer similar to that of wxHaskell. gtk2hs is known to run on Linux, FreeBSD, MacOS X, Windows and Solaris.

4.5.5 HTk

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

HTk is an encapsulation of the graphical user interface toolkit and library Tcl/Tk for the functional programming language Haskell. It allows the creation of high-quality graphical user interfaces within Haskell in a typed, abstract, portable and 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`

4.5.6 HSX11

Report by: *Alastair Reid*
Status: *Maintained, stable*
Last release: *1.00 (6 June 2003)* **Portability:** *GHC, Hugs, Linux, FreeBSD, Solaris, MacOS X*

The Xlib library is a set of bindings to over 300 functions in the standard Xlib C library.

Further reading:

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

4.5.7 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 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 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 Haskell98 + FFI
- No GreenCard 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, 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/`

4.6.2 FunWorlds – Functional Programming and Virtual Worlds

Report by: *Claus Reinke*
Status: *stalled*

FunWorlds is an experiment to investigate language design issues at the borderlines between concurrent systems, animated reactive 2&3d graphics, and functional programming. The only progress over the last year (sic) has been the basic snapshot of the old system I promised in the May 2003 edition. It still is on the wanted list, but don't hold your breath.

Further reading:

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

4.6.3 PanTHeon

Report by: *Sean Seefried*

PanTHeon is re-implementation of Pan, a DSL embedded in Haskell, for the generation of two dimensional images and animations. (Pan was originally developed by Conal Elliott, Oege de Moor and Sigbjorn Finne.)

However this implementation differs from the former in that it is cross-platform and implemented in an entirely new way through Template Haskell (section 3.5). It is built on top of the Simple DirectMedia Layer (SDL) `http://libsdl.org` and has been successfully built on Mac OS X and Linux so far.

Instead of embedding a compiler as the original authors did (mainly for reasons of efficiency), we have opted to use the compile-time meta-programming facilities of Template Haskell to perform domain specific optimizations. This has resulted in an implementation that rivals the speed of the original, while being implemented in far fewer lines of code.

What is its status?

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

Can others get it?

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

What are the immediate plans?

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

4.6.4 Pancito

Report by: *Andrew Cooke*
Status: *updated, not currently being developed further*

A new 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) at `http://www.acooke.org/pancito`

4.7 Web and XML programming

4.7.1 Halipeto

Report by: *Andrew Cooke*

Status: *completed & released, 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 images (section 4.6.4), 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 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 1.11, having recently re-established support for Hugs. Ongoing maintenance is to the CVS tree at haskell.org.

Further reading:

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

4.7.3 Haskell XML Toolbox

Report by: *Uwe Schmidt*
Status: *fourth major release*

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 and HXML, but introduces a more general approach for processing XML with Haskell. The Haskell XML Toolbox uses a generic data model for representing XML documents, including the DTD subset and the document subset, in Haskell. This data model makes it possible to use filter functions as a uniform design of XML processing applications. The whole XML parser including the validator parts was implemented using this design. Libraries with filters and combinators are provided for processing the generic data model.

Features:

- validating XML parser
- very liberal HTML parser
- XPath support
- full Unicode support
- support for XML namespaces
- uniform data model for DTDs and XML content
- native Haskell support of 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 set of example programs have been developed during the last 6 month, to test the toolbox for usability and completeness. This includes a program for generating a Haskell module containing simple access functions and filter for elements and attributes for processing a special XML instance by analyzing a given DTD.
- a prototype of a XSLT implementation is finished, but still needs some extensive testing and integration into the toolbox.
- a project for supporting the Relax NG XML schema definition for validation will start in summer 2004.
- in a second project, also starting in summer 2004, it's planned to write a users guide. In this guide the development of a nontrivial example application will be described, for demonstrating the programming technic 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

New/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 HAIFA

Report by: *Simon Foster*

HAIFA (The Haskell Application Interoperation Framework Architecture) is an experimental interoperability framework for the functional programming language Haskell. When completed, it should enable users to construct interoperating client and server applications with the aim of distributed functional programming over various text-based protocols including for example SOAP and XMLRPC.

HAIFA achieves this with a common call structure, to which calls from the various types of protocols are converted. So for example, in a server environment, HAIFA would convert any incoming SOAP Messages to a HAIFA Call. This involves parsing the messages and mapping the types from the XSD and SOAP type-spaces over to the Haskell type-spaces.

Once a HAIFA Call has been built with Haskell type-names, all the actual data can be unmarshalled (or deserialized), from text to actual typed-data. This data can then be applied to a function or returned (depending on whether HAIFA is working client or server side).

HAIFA serves out applications with HAC, the HAIFA Application Container, which allows several applications working on different protocols to be exposed over a network. HAC has a plugin system, which it uses to dynamically load and unload applications and protocols. HAC communicates with a front-end web-server (such as HWS-WP) for the purpose of exposing the functions over the Internet via HTTP. The eventual aim for HAC is the creation of a generic application server, which can be used to serve out many different kinds of applications including Haskell servlets.

4.7.6 Haskell XML-RPC

Report by: *Björn Bringert*

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/`

Chapter 5

Tools

5.1 Foreign Function Interfacing

5.1.1 GreenCard

Report by:	*Alastair Reid*
Status:	*Maintained, stable*
Last 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:	*beta release*

C–>Haskell is an interface generator that simplifies the development of Haskell bindings to C libraries. It has been ported to Mac OS X and been adapted to GHC's development version 6.3. The tool is currently at version 0.12.0 and has been stress tested in the development of the Gtk+HS GUI library. 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 recently had a 0.3 release: it now 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

Report by:	*Brandon Moore*
Status:	*pre-alpha*

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 accesible at `http://page-208.caltech.edu/phi/`.

5.1.5 HOC: A Haskell to Objective-C binding

Report by: *André Pang*

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

Status: 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: *Simon Marlow*
Status: *stable, maintained*

There have been no new releases of Happy since June 2002. Happy is still in constant use by GHC and other projects, and remains in maintenance mode.

Further reading:

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

5.2.3 HaLex

Report by: *Jorge Sousa Pinta*

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 Pinta*

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 Pinta*

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 (section 4.3.3). Sdf2Haskell is being maintained by Joost Visser (Universidade do Minho, Portugal).

5.2.6 The Utrecht attribute grammar system UAG

Report by: *Doaitse Swierstra*

(Arthur Baars, Doaitse Swierstra)

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

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

Further reading:

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

5.2.7 DrHylo

Report by: *Jorge Sousa Pinta*

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.

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 (section 5.3.3).

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 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/`

5.3.3 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 organization, 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. The refactorings that HaRe supports, including renaming, scope change, generalization and a number of others, are now *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. A snapshot of HaRe is available from our web page.

We continue to develop the system, in dialogue with our users. A workshop in February (details on the web page) allowed us to show the latest features to a group of users and at the same time gather valuable feedback on the existing system. Our major goals for the next few months are to implement data-oriented refactorings and to develop an API which will allow users to write their own refactorings using the utility functions that we have developed. We encourage HaRe-related student projects at other universities and, to avoid overlaps, have started a list of ideas and ongoing or planned projects.

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

5.3.4 VooDooM

Report by: *Jorge Sousa Pinta*

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

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

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 currently only available in a separate branch of GHC in CVS.

5.4.2 Hat

Report by: *Olaf Chitil and Malcolm Wallace*
Status: *static but maintained*

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, we also have an algorithmic debugging viewer (hat-detect) very similar to Buddha, and a forward-animator showing the reduction sequence of expressions (hat-anim). 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, MPTC, fundeps, hierarchical libs).

The most recent public release of Hat (2.02) is now a year old, but since then numerous bugfixes have been applied and several features added in CVS. We intend to release Hat 2.04 at some point in the medium-term future.

Further reading:

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

5.4.3 buddha

Report by: *Bernie Pope*
Status: *active, new release in prepration*

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.

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

Buddha is under active development, with version 1.1.1 having been released since the previous report. Version 1.2 is in preparation. Buddha is freely available as source and is licensed under the GPL.

Version 1.2 will have many new features. The most significant change is the underlying implementation. A totally new program transformation has been developed which will hopefully lead to a better and more efficient debugger.

Other features in 1.2:

- The command line now uses readline.
- ANSI colour is supported.
- The build system has been totally autoconfiscated and automakeified. This means it should be easier to make package distributions of the program (i.e. Debian).
- A Debian package will be provided.
- The C pre-processor is supported.
- Buddha can draw nice diagrams of the callgraph and values that appear in the program using the *dot* graph language.
- An observe command that can show you all the calls to a given function/constant.
- Many other improvements to the interface.

Hierarchical modules are still not supported. Version 1.2 will drop support for version 5 of GHC, unless there is a major public outcry (which I seriously doubt).

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

5.4.4 HUnit

Report by: *Dean Herington*

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 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 (e.g. syntax coloring, context-sensitive help, indication of parse errors while you type), and eventually providing some more advanced features (type checking while you edit, inspecting types of identifiers or subexpressions, refactoring, debugging, GUI tools, etc.).

So far we have basic syntax coloring, dynamic indication of parse errors and even type checking in the editor. Support for projects is currently underway. We plan to work over the summer on finishing the basic features and releasing a beta version.

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>).

5.5.2 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 project has just started (the initial contribution was in March 2004). It follows the example and design principles of the Eclipse project. Every help is very welcome, be it in the form of code contributions (please contact me), docs or tutorials, or just any feedback if you use the IDE. A project site at sourceforge.net has just been created.

There is a working version 0.2 (considered 'alpha') that features a project model, a configurable source code editor (with syntax coloring and Code Assist), compiler support for ghc and launching from the IDE.

Further reading:

`http://eclipse.org`
`http://leiffrenzel.de/eclipse/eclipsefp.html`
`http://eclipsefp.sourceforge.net`

5.5.3 Haddock

Report by: *Simon Marlow*
Status: *stable, maintained, ver. 0.6 released Nov 11, 2003*

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.

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

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

Chapter 6

Applications

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 (section 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 (section 4.4.4) and WASH (section 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. Demo here: `http://shapr.homelinux.net/cgi-bin/wash/SearchML` darcs repo: `http://shapr.homelinux.net/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 the new lambdabot 2.0 release from `http://sf.net/projects/haskell-libs/download`

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*

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

Darcs is still in the process of being stabilized for a 1.0 release. Darcs has seen considerable improvement in the past six months, including performance enhancements, bug fixes and improvements in the user interface. Work on a graphical interface for darcs using wxHaskell has been stalled, although this interface continues to improve as wxHaskell matures.

Darcs is free software licensed under the GNU GPL.
`http://www.abridgegame.org/darcs`

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 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."

Last year Frank Rosemeier has ported Yarrow (written by Jan Zwanenburg in Haskell 1.3, see `http://www.cs.kun.nl/~janz/yarrow/`) to Haskell 98. The Haskell 98 source code has been published on his homepage `http://www.fernuni-hagen.de/MATHEMATIK/ALGGEO/Mitarbeiter/Rosemeier/rosemeierengl.htm`. As this homepage will probably move this year the intention is to offer Yarrow at `http://www.haskell.org/yarrow/`.

The Haskell 98 code will be smoothed by systematically replacing monadic class method notation by do-notation.

Future plans:

- producing documentation with Haddock,
- implementing dependent record types,
- integrating algebraic data types.

6.1.8 HasLaTeX

Report by: *Frank Rosemeier*

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 LaTex2e 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 and probably PPrint), which may be useful for other applications.

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.

For 2004, (2-4 months) I am planning to release

- DoCon-2.08
- Prover: many-sorted term rewriting (TRW) system and inductive prover for predicate calculus, mathematics and programming – based on equational reasoning, TRW ('unfailing' completion and such), and programmed in Haskell

6.1.10 lhs2TEX

Report by: *Andres Löh*

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 has been around since 1997, but was significantly enhanced recently with new features and a manual. It has been used in several papers and documents and seems to be sufficiently stable.

The current version is available at `http://www.cs.uu.nl/~andres/lhs2tex`. The manual, which also serves as a nice example of the capabilities of the tool, can be accessed at `http://www.cs.uu.nl/~andres/lhs2tex/Guide2-1.10pre.pdf`.

6.1.11 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 (section 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`

6.2 Commercial users

6.2.1 Reid Consulting Ltd

Report by: *Alastair Reid*

Status: *still very active*

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

Services we provide:

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

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

Further reading:

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

6.2.2 Aetion Technologies LLC

Report by: *Mark Carroll*

Aetion Technologies LLC continues to use Haskell for most of its in-house development. In past months we have used Functional Reactive Programming for software that attempted to interpret an incoming stream of data. Recently we have been experimenting with Template Haskell using GHC, and with concurrent & distributed computing. 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.

6.3.2 Beseme Project

Report by: *Rex Page*

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

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

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

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

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

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

About two-thirds of the material of the course centers around mathematical logic. After the introduction of predicates, all of the examples in the logic portion of the course involve reasoning about properties of software, most of which is expressed in Haskell (a few are conventional looping functions).

Software examples include sum, sequence concatenation, logical operations on sequences, the Russian peasant algorithm, insertion and lookup in AVL trees, and other computations. Most of the properties verified relate to aspects of program correctness, but resource utilization properties are also verified in some cases.

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

The Beseme website provides access to a preview of the material. Exams and solutions are protected by a login procedure (to increase the comfort level of instructors wishing to use them in courses). To locate the website, just google "Beseme", if it isn't at `http://www.cs.ou.edu/~beseme/`

Chapter 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 semantics 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.

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`

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 tools, like parsers and static analyzers, for languages from the CASL family, in particular CASL itself, HasCASL, and HetCASL, which combines several specification languages such as CSP, CASL, HasCASL, and Modal and Temporal Logic.

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

HTk Graphical User Interfaces for Haskell Programs:
`http://www.informatik.uni-bremen.de/htk`

MMiSS Multimedia instruction in safe systems:
`http://www.mmiss.de`

CASL specification language:
`http://www.informatik.uni-bremen.de/cofi`

Heterogeneous tool set:
`http://www.informatik.uni-bremen.de/cofi/hets`

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 (see also section 2.6.1). Work has progressed on a number of the following projects:

- We are looking at using our port of Haskell (section 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 continuing his work on visual interactive programming with Vital (section 2.6.4) Axel Simon maintains the Gtk2hs binding to the Gtk GUI library (section 4.5.4) and has also been trying to coordinate the Haskell GUI efforts (section 4.5.1). Chris Ryder has evaluated his Metrics and Visualization library Medina through some case studies, and is working on improvements (section 4.3.4 Huiqing Li, Simon Thompson and Claus Reinke have released further snapshots of HaRe, the Haskell Refactorer (section 5.3.3 Olaf Chitil continues improving the Haskell tracer Hat together with the York group (section 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, Robert Pointon, Greg Michaelson, Phil Trinder, Jan Henry Nystrom, 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 (section 3.2.2) and Parallel ML with skeletons (PMLS). Distributed/mobile languages include Glasgow distributed Haskell (section 3.2.3) Erlang (`http://www.erlang.org/`), Hume (`http://www-fp.dcs.st-and.ac.uk/hume/`) 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 is particularly interested in the intersection between theory and practice in language research. Recent highlights concerning Haskell include the following: a proposal for data declarations in type classes to facilitate generic programming for self-optimizing libraries, a plugin library for GHC, and a portable re-implementation of the Pan animation tool using Template Haskell.

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

7.1.8 Institute for Geoinformation at TU Vienna

Report by: *Andrew Frank*

Haskell used for Geographic Information Science Research

We have used Haskell and primarily Hugs for the past 5 years as a language to formalize complex problems in Geographic Information Science:

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

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

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

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

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

Further reading:

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

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

Report by: *Jorge Sousa Pinto*

The Logic and Formal Methods group at the Informatics Department of the University of Minho, Braga, Portugal. `http://www.di.uminho.pt/~glmf`.

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 (section 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 pointfree programming in Haskell (section 4.2.8), 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 (section 5.3.4), 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.

7.1.10 Functional Programming at Utrecht University

Report by: *Arthur van Leeuwen*

All UU Software

(`http://www.cs.uu.nl/groups/ST/`)

We are well on our way to make all our Haskell modules mutually consistent and to make them available through a CVS server at `cvs.cs.uu.nl`, in the directory `uust`. Currently included are our parser combinators, pretty printers, attribute grammar system and a few utilities. Further software will be added. Many of our Ph.D. students are currently working on their Ph.D. theses though, so it may be some time before it is all done.

Parser Combinators

(Doaitse Swierstra, Arthur Baars, Rui Guerra)
See the description of UPC in section 4.3.2.

Helium

(Arjan van IJzendoorn, Bastiaan Heeren, Daan Leijen)
See the description of Helium in section 2.6.2.

Improving Type Errors

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

The attribute grammar system AG

(Arthur Baars, Doaitse Swierstra) See the description of UAG in section 5.2.6.

Utrecht Haskell Compiler

(Atze Dijkstra, Doaitse Swierstra) **Status:** *active development*

The Utrecht Haskell Compiler is an attempt to build a full, working Haskell compiler using the UUST tools. It is not quite ready for consumption yet, but it has already generated running code.

`http://www.cs.uu.nl/groups/ST/Center/UtrechtHaskellCompiler`

Pretty Printing

(Pablo Azero, Doaitse Swierstra)
Our pretty printing combinators have been silently doing their work over the years. Currently we are updating them, so they can be generated by the new version of the AG system. They too will have a more flexible interface allowing naming of subformats by using a monadic top layer.

`http://cvs.cs.uu.nl/cgi-bin/cvsweb.cgi/uust/lib/pprint/`

Proxima

(Martijn Schrage, Johan Jeuring, Lambert Meertens, Doaitse Swierstra)
In the proxima we are designing a layered system for building interactive editors. An interesting aspect of our approach is that we have designed combinators for gluing the different layers that take part in the online formatting. By our knowledge this is the only place where combinators are used to combine really large program structures. We are currently in the state that we can produce small editors in a relatively easy way.

For some screenshots of editors that have been written you may take a look at the projects home page at: `http://www.cs.uu.nl/groups/ST/twiki/bin/view/Center/Proxima`

Syntax Macros

(Arthur Baars, Doaitse Swierstra)
Status: *actively developed*

The syntax macros are now in a state that one gets a macro mechanism for free when using our attribute grammar system and parser combinators in constructing a front end of a compiler. Necessary gluing code is automatically generated. The syntax macros make it possible to extend the context free grammar of a language on a per program basis. Examples of constructs that no longer have to be part of the standard language, but could have been defined us- ing our macro mechanism are the **do**-notation, **arrow**-notation and the notation for list comprehensions. Currently we manage even to give the user feedback in terms of his original program, by allowing online redefinition of the attribute grammar that constitutes the compiler. The latest developments are in trying to incorporate automatic left-factorization into the system, so that the user of the system can't shoot himself in the foot so easily anymore.

The current version is available at `http://www.cs.uu.nl/groups/ST/twiki/bin/view/Center/SyntaxMacros`

First Class Attribute Grammars

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

Generic Haskell

(Johan Jeuring, Andres Löh, Doaitse Swierstra)
See the description in section 3.4.

LiteMECH

(Ade Azurat, Wishnu Prasetya)
LiteMECH is a verification condition generator for a simple imperative programming language. It is implemented in Haskell, using the UUAG system.

7.2 Other groups

7.2.1 Debian Users

Report by: *Isaac Jones*

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

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

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

7.2.2 Fedora Haskell packages

Report by: *Jens Petersen*

In the last edition under "RPM Packaging of Haskell projects" I ended by saying "Also I'm considering contributing some of the major packages like ghc to the Fedora Extras project...". When I turned to do this I discovered that Gerard Milmeister (gemi@bluewin.ch) had already started submitting several core Haskell rpm packages to fedora.us including hugs98 (#842), ghc (#844) and nhc98 (#847). I was hoping to help get them through QA quickly, but unfortunately I got swamped with work, so they are unfortunately *still* waiting in the queue. Getting packages accepted or contributing is really not that hard if you have some experience in RPM packaging and a little determination, and once those packages are accepted it will help to spread Haskell acceptance further and also build the foundation for adding Haskell libraries to Fedora Extras (like gtk2hs (#1519)), so I would encourage people to volunteer a little time to this very worthwhile effort.

Further reading:

project `http://www.fedora.us/` bugzilla `https://bugzilla.fedora.us/` submission process `http://www.fedora.us/wiki/PackageSubmissionQAPolicy`

Chapter 8

Individuals

8.1 Oleg's Mini tutorials and assorted small projects

The page of various Haskell mini-tutorials, type system hacks, and other well-commented mini-projects `http://pobox.com/~oleg/ftp/Haskell/` has received two additions:

- **From enumerators to cursors: turning the left fold inside out**

  The topic of that article (which is a literate Haskell98 code) is traversing collections, e.g., files, databases and generating functions. The article introduces a non-recursive left-fold and argues that it is an optimal traversal interface in a language without first-class continuations. The non-recursive left-fold can be instantiated into either (i) a (recursive) left fold enumerator supporting a premature termination, or (ii) a stream/cursor, which permits safe interleaving. Both instantiation procedures are generic, as evidenced by their polymorphic types. The proposed traversal interface and its two instances (as the enumerator and the cursor) have indeed been implemented, in the TAKUSEN database access library (see section 4.4.3).

- **De-biforestation**

  Another article with accompanying complete Haskell98 code, which considers the problem of eliminating an intermediate data structure and a space leak in a situation where one producer generates two mutually-dependent data streams for two distinct consumers. Furthermore, the rate of production is non-uniform. We derive a deforested version, which no longer needs to buffer any produced items. When we run that version, the memory retaining profile shows no space leaks. We also discuss a "parallel" writing of several streams into two distinct files. Our solution is safe and yet the I/O is effectively interleaved.

8.2 Graham Klyne

Graham Klyne writes:

My primary interest is in RDF (`http://www.w3.org/RDF/`) and Semantic Web (`http://www.w3.org/2001/sw/`) technologies. Since the November HC&A report, I have completed and packaged the first phase of an RDF inference scripting tool, details of which can be found at `http://www.ninebynine.org/RDFNotes/Swish/Intro.html`. I intend that further development is to be driven by specific Semantic Web applications. Identified future work items include:

- a full RDF/XML parser based on the new RDF syntax specification (`http://www.w3.org/TR/rdf-syntax-grammar/`). To this end, I'd like to see the HXml Toolbox fully supported in Hugs under Windows.
- 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 used Haskell to implement and check the revised URI specification, with a view to replacing the current Network.URI module when the specification is stabilized. The current state of my software can be found at `http://www.ninebynine.org/Software/HaskellUtils/Network/`, and the specification can be found at `http://gbiv.com/protocols/uri/rev-2002/rfc2396bis.html`.

Haskell is also used for some smaller utilities, including a report generator compiler, which is being used in conjunction with some Semantic Web tools written in Python to generate IETF message header field registry documentation from RDF source data (e.g. `http://www.ietf.org/internet-drafts/draft-klyne-hdrreg-mail-03.txt`).

I've also published a page of notes about my experience of learning Haskell at `http://www.ninebynine.org/Software/Learning-Haskell-Notes.html`.

Further info on `http://www.ninebynine.org/` or `http://www.ninebynine.net/`.

8.3 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.

I am also about to release its sibling 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.

Everything is, or will be shortly, available from my web pages:

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