Haskell Communities and Activities Report

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

– fourth edition –

May 12, 2003

Claus Reinke (editor), University of Kent, UK
Perry Alexander, The University of Kansas, USA
Krasimir Angelov, Bulgaria
Sengan Baring-Gould, National Semiconductor Corporation
Mark T.B. Carroll, Aetion Technologies LLC, USA
Manuel Chakravarty, University of New South Wales, Australia
Olaf Chitil, The University of York, UK
Matthew Donadio
Joe English, Advanced Rotorcraft Technology, Inc., USA
Levent Erkok, OGI School of Science and Engineering, OHSU, USA
Andre W B Furtado, Federal University of Pernambuco, Brazil
Murray Gross, City University of New York, USA
Jurriaan Hage, Utrecht University, The Netherlands
Keith Hanna, University of Kent, UK
Dean Herington, University of North Carolina at Chapel Hill, USA
Johan Jeuring, Utrecht University, The Netherlands
Ralf Lämmel, VU and CWI, Amsterdam, The Netherlands
Daan Leijen, Utrecht University, The Netherlands
Rita Loogen, University of Marburg, Germany
Christoph Lüth, George Russell, and Christian Maeder, University of Bremen, Germany
Simon Marlow, Microsoft Research Cambridge, UK
Jens Petersen, Red Hat, Japan
John Peterson, Yale University, USA
Henrik Nilsson, Yale University, USA
Rex Page, Oklahoma University, USA
Sven Panne, Germany
Simon Peyton Jones, Microsoft Research Cambridge, UK
Bernie Pope, University of Melbourne, Australia
Alastair Reid, Reid Consulting (UK) Ltd., UK
Chris Ryder, University of Kent, UK
Uwe Schmidt, Fachhochschule Wedel, Germany
Axel Simon, University of Kent, UK
Ganesh Sittampalam, Oxford University, UK
Doaitse Swierstra, Utrecht University, The Netherlands
Anthony Sloane, Macquarie University, Australia
Martin Sulzmann, National University of Singapore, Singapore
Wolfgang Thaller, Graz, Austria
Peter Thiemann, University of Freiburg, Germany
Phil Trinder, Heriot Watt University, Scotland
Eelco Visser, Utrecht University, The Netherlands
Malcolm Wallace, The University of York, UK
Ashley Yakeley, Seattle WA, USA

Preface

Welcome to the fourth edition of our Haskell Communities and Activities Reports! As always, the idea is to give regular snapshots of all things Haskell, including a brief overview of the last 6 months and an outlook of plans for the next 6 months, confirmation of maintainance for existing software, and invitations to contribute to new and ongoing discussions and developments in the numerous communities working on, with or inspired by Haskell.

Editing these reports can be a trying experience at times, so pardon me if I start with some frequently questioned answers that might help to improve the process and ultimately the result: *yes*, we are interested in what **you** are doing with Haskell, especially so if it seems an unusual or new application area. Even to those not working in your particular area, such reports may suggest new inspirations. *Yes*, we do want to hear from maintainers of Haskell software even if there have been no new developments over the last 6 months. A brief confirmation that your software is still actively maintained helps potential users in their choice of tools. *Yes*, it would be much appreciated if everyone could submit their contributions by the end of April (for the May editions) and October (for the November editions). If you leave it late, you're holding up the distribution of information from those who contributed on time.

No, these reports are not about eternal truths. They are about recent events, and if your previous contribution still reads "right" to you 6 months later, it is probably not focussing on where the action is. *No*, a mere pointer to some webpages is not the kind of contribution we have in mind. While these reports are useful prompts for you to undust your webpages, your contributions should complement the webpages by highlighting and summarising recent activites. And *no*, these reports are not half as formal as the end result may sometimes look. It usually takes less time to compose a useful contribution (plain ASCII is best) than it would take to negotiate a late submission in the vain hope of finding time for some grandiose authoring event.

Also, these reports have a wider distribution than the Haskell mailing list where the calls for contributions are posted, and there are many Haskell projects out there from which we have yet to receive summaries of their work. So if you read this, and would like your work to be represented in the next edition, please take out your diary: **submissions to the November edition are due by the end of October**. If you you do not have the time follow the Haskell mailing list closely anymore and would like a personal reminder in mid October, just let me know now and I'll add you to the list!-)

Okay, enough of that, let's look at the contents. The trend towards catering more and more for practical needs is unbroken, and is reflected not only in tools, libraries, and applications. More and more emphasis is also placed on how to organise delivery of Haskell software, how to ensure compatibility, portability, documentation and other aspects of a good user experience, and how to ease maintenance, to make sure that all those goodies are there to stay.

Apart from the ongoing convergence of language and extensions supported by Haskell implementations (2), the hierarchical libraries are being fleshed out and ported, and there is an ongoing discussion about streamlining the processes of contributing and distributing "standard" libraries (4.1). Authors and users of the currently so numerous GUI libraries are not only exchanging views and tips, but have also resumed work towards a common API (4.3.1). Again, it is no longer seen as sufficient to provide any kind of GUI, but it has to be easily portable and maintable, and while platform independence is one way to get there, user expectations about look&feel are of increasing importance.

If one had to pick one area of particular success for Haskell, my bet would be on domain-specific languages, and while most of these come in the form of embeddings in Haskell, there is also a continuous flow of stand-alone implementations (2.5 and 6). And of course, there is steady and encouraging progress in support for parallel, concurrent and distributed programming (3.2).

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

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

Claus Reinke,
University of Kent, UK

Contents

1 General **4**
1.1 Haskell.org . . . 4
1.2 Revised Haskell 98 Report . . . 4

2 Implementations **5**
2.1 The Glasgow Haskell Compiler . . . 5
2.2 Hugs . . . 5
2.3 nhc98 . . . 6
2.4 hmake . . . 6
2.5 Domain-specific variations . . . 6
2.5.1 Haskell on Handheld Devices . . . 7
2.5.2 Helium . . . 7
2.5.3 Educational Domain Specific Languages . . . 7
2.5.4 Vital: Visual Interactive Programming . . . 7

3 Language Extensions **8**
3.1 Foreign Function Interface . . . 8
3.2 Non-sequential Programming . . . 8
3.2.1 Concurrent Haskell . . . 8
3.2.2 GpH – Glasgow Parallel Haskell . . . 8
3.2.3 GdH – Glasgow Distributed Haskell . . . 9
3.2.4 Eden . . . 9
3.3 Type System/Program Analysis . . . 10
3.3.1 Chameleon/A General Type Class Framework based on Constraint Handling Rules . . . 10
3.3.2 Constraint based type inferencing at Utrecht . . . 10
3.4 Generic Programming . . . 11
3.5 Syntactic Sugar . . . 12
3.5.1 Recursive do notation . . . 12

4 Libraries **13**
4.1 Hierarchical Libraries . . . 13
4.2 Data and Control Structures . . . 13
4.2.1 Strafunski . . . 13
4.2.2 DSP Libraries . . . 13
4.2.3 Parsec . . . 14
4.2.4 DData . . . 14
4.2.5 Yampa . . . 14
4.3 Graphical User Interfaces . . . 14
4.3.1 The Common GUI Library Task Force . . . 14
4.3.2 HTk . . . 15
4.3.3 Object I/O for Haskell . . . 15
4.3.4 HToolKit . . . 15
4.3.5 wxHaskell . . . 15
4.3.6 Gtk+HS . . . 16
4.3.7 Gtk2hs . . . 16
4.4 Graphics . . . 16

4.4.1 HGL Graphics Library . . . 16
4.4.2 Win32 and Xlib Libraries . . . 16
4.4.3 HOpenGL – A Haskell Binding for OpenGL and GLUT . . . 16
4.4.4 FunGEn – A game engine for Haskell . . . 17
4.4.5 FunWorlds – Functional Programming and Virtual Worlds . . . 17
4.5 Tool Frameworks . . . 18
4.5.1 Medina – Metrics for Haskell . . . 18
4.6 XML and Web Programming . . . 18
4.6.1 HaXml . . . 18
4.6.2 HXML . . . 18
4.6.3 Haskell XML Toolbox . . . 19
4.6.4 WASH/CGI – Web Authoring System for Haskell . . . 19

5 Tools **20**
5.1 Foreign Function Interface . . . 20
5.1.1 C–>Haskell . . . 20
5.1.2 GreenCard . . . 20
5.1.3 Java VM Bridge . . . 20
5.2 Meta Programming . . . 20
5.2.1 Scanning, Parsing, and Analysis . . . 20
5.2.2 Haskell Transformations . . . 21
5.3 Program Development . . . 22
5.3.1 Tracing and Debugging . . . 22
5.3.2 Refactoring . . . 22
5.3.3 Testing . . . 22
5.3.4 Documentation . . . 23

6 Applications, Groups, and Individuals **24**
6.1 Non-Commercial Applications . . . 24
6.1.1 HScheme . . . 24
6.1.2 Haskell in Alchemy . . . 24
6.1.3 Analysis Tools for Rosetta . . . 25
6.1.4 Hircules, an IRC client . . . 25
6.2 Commercial Applications . . . 25
6.2.1 Reid Consulting Ltd . . . 25
6.2.2 Aetion Technologies LLC . . . 25
6.2.3 Binary Parser . . . 25
6.3 Haskell in Education . . . 26
6.3.1 Beseme Project . . . 26
6.4 Research Groups . . . 26
6.4.1 Formal Methods at Bremen University . . . 26
6.4.2 The Yale Haskell Group . . . 27
6.4.3 Functional Programming at Brooklyn College, City University of New York . . . 27
6.4.4 Functional Programming at Macquarie University . . . 27
6.4.5 Functional Programming at Utrecht University . . . 27
6.4.6 Functional Programming at the University of Kent . . . 29
6.4.7 Programming Languages & Systems at UNSW . . . 29
6.4.8 Parallel and Distributed Functional Languages Research Group at Heriot-Watt University . . . 29
6.5 Individual Haskellers . . . 30

Chapter 1

General

1.1 Haskell.org

Report by: *John Peterson*

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

What can `haskell.org` do for you? There are a lot of things we can do that are of use to members of the haskell community:

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

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

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

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

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

Further reading:

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

1.2 Revised Haskell 98 Report

Report by: *Simon Peyton Jones*

The Haskell 98 Report (Revised) is now finally published, in electronic form at `http://haskell.org/definition`, as Vol 13(1) (Jan 2003) of the Journal of Functional Programming, and as a book published by Cambridge University Press.

The copyright issue was resolved by CUP granting unlimited copying rights, exactly as the previous version of the Report had. This is most unusual for a printed book, and is extremely generous of CUP. I hope you all go out and buy a copy!

I do not propose to make further changes to the Report, but I will continue to accumulate an errata list, so you can send me further bug reports! I have a few already (see the above URL).

Thank you to everyone who contributed. The result is something for us all to be proud of. I gave a "Retrospective on Haskell" talk at POPL'03, which you can find at `http://research.microsoft.com/~simonpj/papers/haskell-retrospective`

Further reading:

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

Chapter 2

Implementations

2.1 The Glasgow Haskell Compiler

Report by: *Simon Peyton-Jones*

The last few months have been mainly consolidation for GHC. We made a couple more releases on the 5.04 branch, which is now very stable, and we are about ready to release GHC 6.0. The main new features in 6.0 will be

1. A new numbering scheme. We think we'll go 6.0, 6.2 etc now, instead of 6.00, 6.02, etc. (In the unlikely event that we do more than five major releases before bumping the major version number, we'll go from 6.8 to 6.10, which seems to be emerging as standard practice.)
2. Template Haskell. We described this in the last Communities Newsletter, and quite a few people are now using it. I'd still describe it as alpha-quality though. More info: `http://research.microsoft.com/~simonpj/papers/meta-haskell/`.
3. Eval/apply. In the last Communities Newsletter we floated a possible change in GHC's basic evaluation model, from push/enter to eval/apply. After quite a bit of experimentation, we decided to commit this change, which will be in 6.0. It's not a programmer-visible change, but it does make code generation and the runtime system rather easier to deal with. We made lots of measurements and submitted a paper to ICFP (`http://research.microsoft.com/~simonpj/papers/eval-apply`).

There are several other projects on the go:

1. Robert Ennals, a Cambridge research student, has a cunning scheme called *optimistic evaluation*, which aims to use call-by-value instead of call-by-need by default, with an abortion mechanism to ensure that the semantics of the program is unchanged. He's implemented this idea in GHC (twice!) and gets very encouraging results: many programs go much faster (doubling in speed is not unusual), while none slow down significantly. There's a paper at `http://research.microsoft.com/~simonpj/papers/optimistic/`.

   We will almost certainly fold this into the main GHC development trunk sometime in the next few months.
2. Umut Acar, a CMU research student, has been visiting for a few months, and is implementing a system of polymorphic records for GHC. The design is at `http://research.microsoft.com/~simonpj/Haskell/records.html`; it's a bit less powerful than TRex, and easier to implement. We're not sure how records should develop in Haskell – it seems impossible to avoid conflict with the existing record syntax – but this is one attempt to make progress.
3. The "Scrap your boilerplate" approach to generic programming in Haskell looks pretty promising (`http://research.microsoft.com/~simonpj/papers/hmap/`) so we are in the process of implementing `deriving(Typeable)` and `deriving(Data)`, so that it's both less painful and more efficient to use these classes. (The `Data` class is called `Term` in the paper.) We're considering removing the derivable-type-class extension, which few people are using, but we'll wait to see if there are any applications that the old approach can handle but the new one can't.

Further reading:

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

2.2 Hugs

Report by: *Alastair Reid*
Project status: *Actively maintained, stable*

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.

Team / status

The Hugs98 interpreter is now maintained by Sigbjorn Finne and Jeffrey Lewis, both of Galois Connections, with help from Alastair Reid of Reid Consulting and Ross Paterson of City University London and others.
The last major release (November 2002) greatly improved compatibility between Hugs and GHC by providing a significant subset of GHC's hierarchical libraries and adding the standard FFI interface extension.

Future plans

Since the last release, Hugs has undergone an internal cleanup to aid future development: long-dead features are being removed and internals are being reorganized. As promised in the previous report, future releases will not have as much backwards compatability as the last release. Ross Paterson is adding many of the hierarchical libraries omitted from the last release. We are also making moves to create a new users guide which documents all of Hugs' extensions and recent additions.

Further reading:

http://www.haskell.org/hugs/
http://haskell.org/mailman/listinfo/hugs-users/

2.3 nhc98

Report by: *Malcolm Wallace*
nhc98 is a small, easy to install, standards-compliant compiler for Haskell 98, the lazy functional programming language. It is very portable, and aims to produce small executables that run in small amounts of memory. It produces medium-fast code, and compilation is itself quite fast. It also comes with extensive tool support for automatic compilation, foreign language interfacing, heap and time profiling, tracing, and debugging.

Recent news

The latest release of nhc98 is 1.16, available since early March, and its features of note are:

- A large subset of the 'base' package of hierarchical libraries is now included in the build.
- The primitive FFI mechanism has been updated to match the latest official spec, and the full Foreign libraries are also included (in hierarchical form).
- nhc98 once again builds on Windows (Cygwin) with ghc.
- nhc98 now works correctly in the presence of gcc-3.x.
- The library function List.sortBy now uses a stable O(n log n) mergesort.
- Numerous other small fixes, including revisions to the Haskell'98 standard.
- The website is now hosted at haskell.org
- Our CVS repository is also now hosted at cvs.haskell.org

Future plans

The nhc98 compiler is pretty stable and reliable and we don't have any particular plans for new features. Bugfixes will appear as and when necessary. However, you users may have ideas for exciting things to add into the compiler, and you are encouraged to get your hands dirty, implement them, and submit them for distribution.

Further reading:

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

2.4 hmake

Report by: *Malcolm Wallace*
Hmake is an intelligent compilation management tool for Haskell programs. It automatically extracts dependencies between source modules, and issues the appropriate compiler commands to rebuild only those that have changed, given just the name of the program or module that you want to build.

Recent news

The latest release of hmake is 3.07, available since early March. This fresh version has the following improved configuration features over previous releases:

- Hmake now once again builds cleanly with GHC under Cygwin.
- Better handling of config files. Your personal config file is now used as an override for the system-wide config file, rather than being used instead of it. Hence, any global config updates are now automatically propagated to all users.
- The new command 'hmake-config new' is now needed to begin a fresh personal config file, rather than one being created for you silently against your expectations.
- 'hmake-config list' can now take a specific compiler argument to show detailed info for that compiler.

Future plans

It has been suggested that hmake should allow the external configuration of different source code preprocessors in a similar manner to the way it currently supports different compilers. This sounds like a useful idea, which we will probably introduce sometime soon.

Further reading:

http://www.haskell.org/hmake

2.5 Domain-specific variations

In addition to the well known major Haskell implementations, there are now several domain-specific implementations, where suitability for the intended application domain is deemed more important than full support for Haskell 98, or where indeed the language subset and design modifications have been explicitly tailored to the domain..

2.5.1 Haskell on Handheld Devices

Report by: *Anthony Sloane*

In 2002 one of our honours students, Matthew Tarnawsky, completed an initial port of the nhc98 runtime to Palm OS. The port is only really of alpha status, but simple programs can be compiled and linked on a desktop machine, installed on a Palm and executed. Limited support for interfacing with the Palm OS GUI libraries is included.

Our current work (cf. section 6.4.4) involves redoing the implementation to bring it up to the latest release of nhc98, reconsidering some of the design decisions made during Matthew's project, and improving the integration with Palm OS. We hope to have a beta version and a report ready for submission to this year's Haskell workshop.

2.5.2 Helium

Report by: *Daan Leijen*

(Arjan van IJzendoorn, Bastiaan Heeren, Daan Leijen, Rijk-Jan van Haaften)

The purpose of the Helium project is to construct a light-weight compiler for a subset of Haskell that is especially directed to beginning programmers. 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. Helium now has a simple graphical user interface that provides online help. We plan to extend this interface to a full fledged learning environment for Haskell. The complete type checker and code generator has been constructed with the attribute grammar (AG) system developed at Utrecht University (section 6.4.5). One of the aspects of the compiler is that it also logs errors, so we can track the kind of problems students are having, and improve the error messages and hints. The compiler uses Daan Leijen's LVM (Lazy Virtual Machine) as back-end. The LVM uses a portable instruction set and file format that is specifically designed to execute lazy higher-order languages.

Further reading:

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

2.5.3 Educational Domain Specific Languages

Report by: *John Peterson*

Project status: *maintained, stable*

The goal of this project is to bring functional programming to users that are not trained computer scientists or programmers. We feel that the simplicity of functional programmiung makes it an ideal way to introduce programming language concepts to students of all ages. We also believe that domain specific languages based on Haskell can enhance learning in other domains such as mathematics.

Our languages are media oriented. They allow students to explore the basic principles of functional programming while creating images, animations, or 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#) is capable of handling all of the examples in Conal Elliott's Fun of Programming chapter.

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 very easy to install and use. This probably would run on Linux using Mono (.NET for other platforms) but we have not attempted this yet. The front end of this system is a mini-Haskell interpreter which is currently somewhat unsophisticated - we plan to customize Helium (section 2.5.2) for this purpose in a future release. Our website contains a number of examples produced by this language and some instructional materials.

Our second language describes music using Paul Hudak's Haskore system. We are currently re-packaging Haskore using Helium to make the system more student friendly.

Further reading:

`http://haskell.org/edsl`

2.5.4 Vital: Visual Interactive Programming

Report by: *Keith Hanna*

Project status: *on-going*

Vital is a Haskell-based, visual environment intended for the interactive, exploratory development of programs by non computer-specialist end-users (engineers, analysts, etc.).

In the environment, each Haskell module is associated with a worksheet on which its declarations and expressions may be located and their values graphically displayed (in a form determined by a type-indexed stylesheet).

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

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

Further reading:

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

Chapter 3

Language Extensions

3.1 Foreign Function Interface

Report by: *Manuel Chakravarty*
Project status: *Version 1.0 (RC8)*
The Haskell 98 FFI Addendum is meanwhile up to Release Candidate 8, which resolves the issues surrounding finalizers for foreign objects that was mentioned in the previous HC&A Report. The current interface definition is available from
`http://www.cse.unsw.edu.au/~chak/haskell/ffi/`

Further reading:

`http://haskell.org/mailman/listinfo/ffi/`

3.2 Non-sequential Programming

3.2.1 Concurrent Haskell

Report by: *Simon Marlow*
Project status: *maintained, stable*
Concurrent Haskell is a set of extensions to Haskell to support concurrent programming. The concurrency API (Concurrent) has been stable for some time, and is supported in two forms: with a preemptive implementation in GHC, and a non-preemptive implementation in Hugs. The Concurrent API is described here:
`http://www.haskell.org/ghc/docs/latest/html/base/Control.Concurrent.html`
There is an ongoing discussion to decide how Concurrent Haskell should work in a system with OS-level threading; in particular what minimal support is required to give the programmer enough control over the correspondence between OS-level threads and Haskell threads in order to use some existing foreign-language libraries that make use of OS thread-local state. The discussion is taking place on the FFI list; see
`http://www.haskell.org/pipermail/ffi/`
Wolfgang Thaller summarizes: All current implementations of Concurrent Haskell do their own scheduling; none currently use the thread functionality provided by the OS; instead, they execute all Concurrent Haskell threads in one OS thread. This has some advantages (much better performance, for example), but it also causes problems. Some foreign libraries (most notably OpenGL) rely on using thread-local state for their interface. Using these libraries from multiple Haskell threads leads to major confusion. These libraries can currently only be used from a single thread by Haskell programs, or sometimes even not at all. Over the past months, I've been "annoying" people on the FFI mailing list by posting various proposals on how to deal with this problem. The general idea is to "bind" some Haskell threads to specific OS threads; all foreign calls from those threads are guaranteed to be executed in a dedicated OS thread, so that thread-local state causes no problems anymore. I'm also planning to implement the proposal that we'll finally agree on for GHC, hopefully before the next major release.

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, Murray Gross, Andre Rauber du Bois, Alvaro Rebon Portillo, Leonid Timochouk.*

Status: A complete, GHC-based implementation of the parallel Haskell extension GpH and of evaluation strategies is available. Extensions of the runtime-system and language modules, to improve performance and support for architecture-independence, are under development.

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

- The *stable branch* (GUM-4.06.2, based on GHC-4.06) is available for RedHat-based Linux machines: as source bundle and as a binary snapshot for RedHat 8.0. A slightly older version GUM-4.06 is available as source bundle or binary snapshot for RedHat 7.0 Intel Linux (libc2.1). See installation instructions. The stable branch is available from the GHC CVS repository via tag gum-4-06.
- The *unstable branch* (GUM-5.02, based on GHC-5.02) is currently being tested on a Beowulf cluster. Most of our test programs run already, with minor issues left to be resolved before this version will become our main development version. The unstable branch is available from the GHC CVS repository via tag gum-5-02-3.

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

- A port to a Sun-Solaris shared-memory machine exists but currently suffers from performance problems, which we are trying to track down.
- A port to an SGI-Irix multi-processor is underway at Universidad Complutense de Madrid.
- A port to a Mosix cluster is being built in the Metis project at Brooklyn College (section 6.4.3), with a first version available on request from Murray Gross.

System Evaluation and Enhancement:

- We have undertaken a comparison of implementation designs for parallel functional languages (`http://www.macs.hw.ac.uk/~dsg/gph/papers/drafts/ppdp.ps.gz`), specifically GUM implementation of GpH with the Skeleton-based parallel ML implementation PMLS (`http://www.cee.hw.ac.uk/Research/funct_prog.html`)
- We have evaluated the effect of introducing thread migration (`http://www.macs.hw.ac.uk/~dsg/gph/papers/ps/Migration-IFL02.ps.gz`) in GUM (IFL02).
- We are currently investigating the challenges posed by porting GUM to a computational GRID, replacing the current PVM layer with IMPICH-G2.
- We are starting to investigate engineering a combined, modular Eden (section 3.2.4)/GpH implementation

GpH Applications:

- A new EPSRC project (GR/R91298) has just started at St Andrews University to investigate providing parallel implementations of the GAP group algebra libraries using GpH.
- We have investigated the architecture independence (`http://www.macs.hw.ac.uk/~dsg/gph/papers/drafts/hlpp03.ps.gz`) of GpH by measuring a significant application (genetic alignment) on two very different architectures: a Beowulf cluster, a SunServer SMP.

Language: We are constructing efficient implementations of algorithmic skeletons in GdH, and plan to experiment with their use, in conjunction with evaluation strategies.

Further reading:

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

3.2.3 GdH – Glasgow Distributed Haskell

Report by: *Phil Trinder*

The Team: *Phil Trinder, Hans-Wolfgang Loidl, Jan Henry Nystrom, 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* (GR/R88137) has just started at Heriot-Watt University to evaluate GdH and Erlang in a telecommunications context (`http://www.cee.hw.ac.uk/~dsg/telecoms/`). The project is in conjunction with Motorola UK Research Labs.
- A GdH prototype has been constructed of a proposed Haskell extension to support mobility(WFLP'03; `http://www.macs.hw.ac.uk/~trinder/papers/mHaskellDesign.ps`).
- GdH is being used to construct efficient implementations of algorithmic skeletons for use in parallel GpH programs.
- GdH and Eden (section 3.2.4) are being compared, based on a distributed file server constructed in both.

Further reading:

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

3.2.4 Eden

Report by: *Rita Loogen*

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:

Madrid:	*Ricardo Peña, Yolanda Ortega-Mallén, Mercedes Hidalgo, Rafael Martinez, Clara Segura*
Marburg:	*Rita Loogen, Jost Berthold, Steffen Priebe*

Eden extends Haskell by a small set of syntactic constructs for explicit process specification and creation. While providing enough control to implement parallel algorithms efficiently it frees the programmer from the tedious task of managing low-level details by introducing automatic communication (via head-strict lazy lists), synchronisation, and process handling. Eden's main constructs are process abstractions and process instantiations. The expression `process x -> e` of a predefined polymorphic type `Process a b` defines a *process abstraction* mapping an argument `x::a` to a result expression

`e::b`. Process abstractions of type `Process a b` can be compared to functions of type `a -> b`, the main difference being that the former, when instantiated, are executed in parallel. *Process instantiation* is achieved by using the predefined infix operator `(#) :: Process a b -> a -> b`.
Higher-level coordination is achieved by defining higher-order functions over these basic constructs. Such *skeletons*, ranging from a simple parallel map to sophisticated replicated-worker schemes, have been used to parallelise a set of non-trivial benchmark programs.
Eden has been implemented by modifying the parallel runtime system GUM of GpH (section 3.2.2). Differences include stepping back from a global heap to a set of local heaps to reduce system message traffic and to avoid global garbage collection. The Eden runtime system has recently been restructured to exhibit a layered structure. The main idea has been to specify the process control in Haskell and to restrict the extensions of the low-level runtime system to a few selected primitive operations. Details can be found in the forthcoming Euro-Par paper *"High-level Process Control in Eden"*. 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.

Current activities:

- *Yolanda* and *Mercedes* are working on the denotational semantics for Eden. They have already developed a continuation-based model for process creation and single-value communication. Currently they are continuing work in three directions:
  1. extending the model to deal with streams and non-determinism;
  2. relating the model to their operational semantics;
  3. applying their continuation-based model to other parallel functional languages, namely, pH and GpH.
- *Ricardo* and *Clara* have just finished the correctness proof of their non-determinism analysis with respect to a simplified denotational semantics for Eden.
- *Raphael* and *Ricardo* are porting the Eden compiler to the IRIX-MIPS multiprocessor platform. Moreover, they are developing an interface between Eden and the computer algebra system Maple. Thus, it will soon be possible to run computation-intensive algorithms from the algebra field in parallel.
- *Steffen* is working on analysis techniques and a polytypic skeleton library.
- *Jost* has incorporated several optimisations into the Eden runtime system: large data will automatically be split and sent in several messages while small pieces of data which have to be sent to the same processor element will automatically be collected into one message to reduce communication costs.
- *Kevin Hammond*, *Jost* and *Rita* are investigating the use of Template Haskell for automatically selecting appropriate skeleton implementations at compile-time.
- *Rita* and *Jost* plan to develop a generic parallel runtime system that can support multiple high-level languages and that offers implicit control of key runtime aspects such as thread management, synchronisation and communication. First, an aspect-oriented generic toolkit for high-level parallel computations will be built.

Further reading:

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

3.3 Type System/Program Analysis

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

Report by: *Martin Sulzmann*
Project status: *on-going*

We use Constraint Handling Rules (CHRs) to describe various type class extensions. Under sufficient conditions on the set of CHRs, we have decidable operational checks which enable type inference and ambiguity checking for type class systems. We have incorporated the ideas of the CHR-based overloading approach into an actual programming language called Chameleon. The syntax of Chameleon follows mostly Haskell. We plan to use Chameleon as an experimental test-bed for possible type system extensions. Recent developments:

- Chameleon now comes with a declarative type debugging interface.
- We can handle a significant subset of Haskell.
- As an undocumented feature, the latest release compiles Chameleon to plain Haskell (via the evidence translation scheme). Some proper documentation will follow shortly.

Further reading:

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

3.3.2 Constraint based type inferencing at Utrecht

Report by: *Jurriaan Hage*
Project status: *on-going*

Participants: *Jurriaan Hage, Bastiaan Heeren, Doaitse Swierstra*; all from Universiteit Utrecht.
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 (section 2.5.2) compiler developed at Universiteit Utrecht.
The main characteristics of the inferencer are the following.

- 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.
- A major innovation is the ability for a Helium programmer to develop his own type rules for any combinator library he might be writing. In addition to having a large degree of control over the type inference process, the system allows a programmer to generate domain specific error messages and specify that certain functions are often mixed up (the compiler may for instance give the hint that (++) should be used instead of (:), because (++) happens to fit in the context). The user defined 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.

At this point, the Helium compiler does not include type classes, records and a few other elements of Haskell. Type classes shall be included in one form or another on short notice. Our type inferencer shall be extended accordingly.

Further reading:

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

3.4 Generic Programming

Report by: *Johan Jeuring*

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

Preprocessors DrIFT is a preprocessor which generates instances of generic functions. It is used in Strafunski (section 4.2.1) to generate a framework for generic programming on terms.

Languages Light-weight generic programming: Generic functions for data type traversals can (almost) be written in Haskell itself, as shown by Ralf Lämmel and Simon Peyton Jones in *'Scrap your boilerplate'* (`http://research.microsoft.com/Users/simonpj/papers/hmap/`). The scrap your boilerplate ideas are currently being implemented in GHC, using the deriving construct. In *'Strategic polymorphism requires just two combinators!'* (`http://www.cwi.nl/~ralf/ifl02/`), Ralf Lämmel further develops these ideas. Another light-weight approach, using type representations inside Haskell, was presented by Cheney and Hinze at the Haskell workshop, and further elaborated upon by Hinze in Gibbons and de Moor (editors), *The Fun of Programming*, Palgrave, 2003. Generic programs can also be implemented in a language with dependent types, as shown by McBride and Altenkirch in a paper in WCGP'02, see `http://www.dur.ac.uk/c.t.mcbride/generic/`. More about generic programming and type theory (*'Generic Haskell in type theory'*) can be found in Ulf Norells recent MSc thesis `http://www.cs.chalmers.se/~ulfn`.
The Generic Haskell release of last summer supports type-indexed data types, dependencies between generic functions, and special cases for constructors (besides the 'standard' type-indexed functions and kind-indexed types). These extensions are described in the *"Type-indexed data types"* paper presented at MPC'02, and the *"Generic Haskell, Specifically"* paper at WCGP'02. The new Generic Haskell release was used in the Summer School in Generic Programming in Oxford last August, at which Ralf Hinze and Johan Jeuring presented two tutorials: *Generic Haskell - theory and practice*, and *Generic Haskell - applications*. The former tutorial introduces Generic Haskell, and gives some small examples, the latter tutorial discusses larger applications such as an XML compressor. The tutorials will appear in LNCS.
A small group at Chalmers is working under the slogan *"Functional Generic Programming - where type theory meets functional programming"* (`http://www.cs.chalmers.se/~patrikj/poly/`), and has written a paper on different universes of codes for types (avaliable upon request). PolyP is not really actively developed anymore, but a new version using a class-based translation is available upon request.
Roland Backhouse and Jeremy Gibbons will start a project on datatype-generic programming in August 2003. The goal of this project is, amongst others, to develop a methodology for constructing generic programs.

Current Hot Topics: 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. Immediately after the next release of Generic Haskell we will start adding type checking and inferencing to the dependency-style Generic Haskell compiler, and we will implement explicitly recursive generic functions. Other: the relation between generic programming and dependently typed programming; the relation between generic programming and Template Haskell (which in prototype form has been implemented in GHC, email <template-haskell@haskell.org> to gather feedback, and see section 2.1); methods for constructing generic programs.

Major Goals: Next release of Generic Haskell: very soon, before summer 2003. Use generic programming to provide a data binding from XML schema to (Generic) Haskell. 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/`
There is a mailing list for Generic Haskell: <generic-haskell@cs.uu.nl>. See the homepage for how to join.

3.5 Syntactic Sugar

3.5.1 Recursive do notation

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

People: *Levent Erkok, John Launchbury*
The recursive do-notation (a.k.a. the mdo-notation) is supported by all Hugs releases since February'01. For GHC (as of 5.04.3), only the latest CVS version supports mdo. (The next major release for GHC will include direct support.) Both implementations are stable and actively supported.

Further reading:

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

Chapter 4

Libraries

4.1 Hierarchical Libraries

Report by: *Simon Marlow*

Portability of the hierarchical libraries is improving: Hugs has shipped a new version with a substantial subset of the hierarchical libraries, and the NHC is rumoured to work with some of these libraries too.

The hierarchical replacement for the old Posix library is about 95% complete, and will ship with the next release of GHC.

An important issue that remains to be addressed satisfactorily is the means by which implementations of libraries are contributed to the community. A discussion on this subject was recently started on the Haskell mailing list: `http://www.haskell.org/pipermail/haskell/2003-April/011624.html` If you have opinions or ideas, please contribute to the discussion.

Contributions of documentation for libraries are still needed: many of the implementations of "standard" libraries that originated in the Haskell 98 language & library reports are poorly documented, and in many cases not documented at all beyond the basic documentation that Haddock extracts from the source.

Further reading:

`http://www.haskell.org/~simonmar/libraries/libraries.html`
`http://www.haskell.org/~simonmar/lib-hierarchy.html`
`http://www.haskell.org/mailman/listinfo/libraries/`

4.2 Data and Control Structures

4.2.1 Strafunski

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

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

Recent papers on Strafunski cover application domains such as Cobol reverse engineering, Java software metrics, and Haskell reengineering:

- A Strafunski application letter (PADL 2003) `http://www.cwi.nl/~ralf/padl03/`
- A Framework For Datatype Transformations (LDTA 2003) `http://www.cwi.nl/~ralf/fdt/`

The current Strafunski release followed closely the previous edition of this HC&A report (November 2002). Strafunski bundles now 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.

Further reading:

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

4.2.2 DSP Libraries

Report by: *Matthew Donadio*
Project status: *resurrected, active, maintained*

The Haskell DSP library has grown by leaps and bounds since I resurrected the project a few months ago. The webpage hasn't been updated in a few weeks, but represents what functions are available.

In particular, there is an FFT library that works for real and complex data, and for all sizes sizes of N. It is also designed to make it easy for users to experiment with gluing together the algorithms in their own way. There are functions for basic DSP functions such as correlation, convolution, FIR filtering, and IIR filtering. Work has begun on adding functions for both FIR and IIR filter design. There are also algorithms for frequency and spectral estimation.

Some of the modules have broader applications than just DSP. I will be soliciting comments and opinions on this on the libraries list in a few weeks.

There are some bits and pieces that rely on GHC extensions, otherwise, it should work with all Haskell98 implementations. While still experimental, it is currently usable. The library was used to verify methods and generate data for an article submitted to the IEEE Signal Processing Magazine.
Work is underway to migrate everything to a hierarchical library, add Haddock style documentation, and provide more demo applications. Be warned, though, that the API is experimental and will likely change.
If anyone uses the library, please let me know what you think.

Further reading:

`http://users.snip.net/~donadio/haskell/`

4.2.3 Parsec

Report by: *Daan Leijen*
Parsec is a practical parser combinator library for Haskell that is well documented, has extensive libraries, and good error messages. It is currently part of the standard Haskell libraries and, fortunately, no new bugs have been reported for a while now. We plan to update Parsec this summer by fixing some naming issues within the new hierarchical namespace, bringing the standalone version in sync with the GHC version, and by adding a module that adds combinators to parse according to the (full) Haskell layout rule.

Further reading:

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

4.2.4 DData

Report by: *Daan Leijen*
DData is a library of efficient data structures and algorithms for Haskell (Set, Bag, and Map). I noticed that I used these data structures (and algorithms) a lot, but surprisingly, there are not many "off the shelf" libraries available. The modules are extensively documented and are not dependent on each other or other support modules. The modules are implemented to be easy to use (just import it), efficient (use the best known implementation techniques), and fairly complete (lots of operations).

Further reading:

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

4.2.5 Yampa

Report by: *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 a preliminary version of Yampa that contains:

- The **Yampa Base Library**, containing generic functions for the expression of signal functions operating on continuous as well as discrete signals, and advanced switching constructs for the interaction between the continuous and discrete worlds.
- The **Yampa Robotics Library**, containing entities tailored for controlling mobile robots, both real and simulated, in the style of Frob, our FRP-based robotics language. The simulator is written using Yampa's Base and HGL, the Haskell Graphics Library (section 4.4.1), and performs physical modelling of mobile differential-drive robots equipped with several kinds of sensors. A preconfigured version of the simulator allows one to play RoboCup Soccer.
- A tutorial (from the *2002 Summer School on Advanced Functional Programming*, Oxford, UK).

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

Further reading:

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

4.3 Graphical User Interfaces

Haskell continues to attract new approaches to GUI library provision. If all past Haskell GUI libraries were still in existence, and still maintained, and available on all platforms, one could easily use a different one for each window. Unfortunately most of them aren't, aren't, and never were.
Currently, we seem to have entered a new era of GUI diversity, and it is encouraging that so many Haskellers are able to create new GUI bindings from scratch when the existing ones do not suit their needs (in addition to those listed here, see also Mike Thomas' entry on Japi in section 6.5).
Even more encouraging, however, is that todays GUI developers are very much concerned with practical issues (how easy is it to create and maintain a GUI library, how easy is it to port to the major platforms, what GUI library users want). Difficulties encountered during attempts to port the otherwise very promising ObjectIO to Gtk have also restarted the discussions and activities towards a common, platform independent GUI API on the gui mailing list.

4.3.1 The Common GUI Library Task Force

Report by: *Axel Simon*
Project status: *revived*
Haskell suffers from too many half-baked GUI libraries. This task force set out to define a common GUI API (CGA for

short) that defines a portable interface for all major platforms. A discussion early this year resulted in the following requirements:

- Platform independence: Programs written according to the Common GUI API should run on all platforms.
- Look-and-Feel: The CGA will only provide functionality that maps to all supported platforms without violating the native style guide. Thus the CGA will exhibit some kind of subset of all platforms. The fixed set of supported platforms is as follows: Win32, Mac OS X, Gnome (Gtk), X11/Xt (Motif/Athena).
- Extensibility: Additional functionality that cannot be expressed with the native look-and-feel can and should be provided by the different back-ends.
- Medium abstraction level: The CGA will not impose any functional model but an imperative interface on which high-level approaches can be based.

The discussion is open to everyone. We discuss on the <gui@haskell.org> mailing list. Documents we create are available in the Haskell CVS repository at `http://cvs.haskell.org/cgi-bin/cvsweb.cgi/haskell-report/gui/`.

Further reading:

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

4.3.2 HTk

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

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 manner. HTk is known to run under Linux, Solaris, Windows 98, Windows 2k, and will probably run under many other POSIX systems as well. It works with GHC, versions 5.02.3 and later.

Further reading:

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

4.3.3 Object I/O for Haskell

Project status: *completed, but no longer maintained*

The Object I/O is a flexible library for building rich user interfaces. It is a port of the popular Clean Object I/O to Haskell (`http://www.cs.kun.nl/~clean/`). The current implementation for Clean and Haskell supports only the Windows platform. Attempts to port the Haskell version to Gtk turned out to be difficult, highlighting platform-specific assumptions in the design, and prompting a renewed discussion on portable GUI APIs on the gui mailing list.

While the implementation is still available, it is no longer developed or maintained (Krasimir is now working on a successor, HToolKit, see below).

Further reading:

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

4.3.4 HToolKit

Report by: *Krasimir Angelov*

The HToolkit is the direct descendant of the ObjectIO. However they are very different. A certain time after I have completed the port of ObjectIO from Clean to Haskell, I was working on the port of Object IO to Linux. Unfortunately the task was harder than I thought. The main difficulty is that the Linux port requires lots of changes on low levels, which sometimes affect the higher-level interface. To solve this difficulty I decided to start the new HToolkit Project. The main objectives are:

1. Development of platform independent GUI library.
2. The library should use only native libraries and where possible native controls. Neither Qt nor GTK use native controls. This means that the applications would look different on different platforms (following the native look and feel as far as possible).
3. The OS dependent layer of the library should be written in C. This will allow the library to be used by other languages (SmallTalk, Python, Eiffel). Two months ago Daan Leijen offered to separate the OS dependent layer as a separate library, independent of Htoolkit. The new library was named **Port**. This change will allow to experiment with libraries of higher level with various designs by using one and the same OS layer. On top of Port Daan built **GIO** Library. Its design follows the design of Yahu. For the time being Daan does not support GIO but I decided to continue GIO/Port development. The most peculiar about GIO is the way to get/set the attributes of graphical components. From the discussion in `gui@haskell.org` I am under the impression that the idea is widely accepted from the community. I don't know how "Common GUI API (CGA)" will look like (discussed in `gui@haskell.org`). Probably the GIO interface will be rather different from CGA but I hope that the efforts put into Port should be reused in CGA.

Further reading:

`http://htoolkit.sourceforge.net`

4.3.5 wxHaskell

Report by: *Daan Leijen*
Project status: *alpha*

wxHaskell is a Haskell binding to the wxWindows GUI library. The wxWindows library is a large C++ library that is portable to all major GUI platforms - Windows, GTK, Motif, X11, MacOS 9, and MacOS X. Furthermore, it is a mature library (in development since 1992) that supports a wide range of widgets with a native look-and-feel.

We hope that this "industrial strength" library can serve as the standard low-level GUI library for Haskell. By using a standard library like wxWindows we will be able to support a wide range of platforms and widgets with almost no effort on our side – we just have to keep in sync with the library. Note that the languages Eiffel, Perl, and Python have also chosen this library for writing complex graphical user interfaces (wxEiffel, wxPerl, and wxPython respectively).
The "core" Haskell binding to wxWindows is generated automatically from the wxEiffel binding and contains about 2500 methods in 500 classes with 1000 constant definitions. As such, most of the wxWindows functionality is supported, just excluding more exotic widgets like dockable windows and the OpenGL canvas. On top of the core binding, we are writing a small library that offers a somewhat more friendly interface to the raw wxWindows interface. This library is closely structured after GIO and Yahu.
The project is in its startup phase - no binary releases have been made and only development is supported. The library has currently only been tested on win32 and GTK (Linux) platforms with GHC 5.03+ and only small examples have been written. We plan to release the initial alpha release in May 2003.

Further reading:

`http://wxhaskell.sourceforge.net`

4.3.6 Gtk+HS

Report by: *Manuel Chakravarty*
Project status: *beta release*
Gtk+HS is a Haskell binding to the GTK+ GUI toolkit `http://www.gtk.org/`, which is currently at version 0.15.2. Since the last HC&A Report, it has acquired a range of new widgets as well as significantly extended support of the **iHaskell** layer, which enables GUI programming without mutable variables.

Further reading:

More details as well as source and binaries packages are at `http://www.cse.unsw.edu.au/~chak/haskell/gtk/`

4.3.7 Gtk2hs

Report by: *Axel Simon*
Project status: *beta*
Gtk2hs is a wrapper around the latest Gtk release (Version 2.2 or Gtk 2 for short). Although it provides a similar low level veneer like Gtk+HS, it is completely rewritten from scratch, reflecting the new object hierarchy of Gtk 2. The binding works on Unix, MacOS X and Windows (mingw32). A thin wrapper called **MOGUL** (MOnadic GUi Library) provides some convenience functions and will be extended to exhibit the Common GUI API once it is defined. All widgets are bound, although a couple of functions are still missing. Future additions will include the OpenGL widget GtkGLExt and the Pixbuf image manipulation library.
Our current work is done in a CVS repository which can be found on `http://sourceforge.net/projects/gtk2hs/`.

4.4 Graphics

4.4.1 HGL Graphics Library

Report by: *Alastair Reid*
Project status: *Maintained, stable*
Portability: *GHC, Hugs, Linux, FreeBSD, Solaris, MacOS X, Windows*
The HGL provides an easy to use, portable interface to Win32 and X11 which supports simple 2-dimensional graphics, keyboard, mouse and timer input events and multiple windows. The library is distributed as open source and is suitable for use in teaching and in applications.
Status: The library works on both Win32 and X11 under Hugs and GHC. The API is stable and the library is used throughout Paul Hudak's 'School of Expression' textbook. The last release was 2.0.4 in December 2001. A release that works better with the new release of Hugs (notably the support for hierarchical module namespace) and current releases of GHC will be available soon.

Further reading:

HGL web page:	`http://haskell.org/graphics/`
School of Expression web page:	`http://haskell.org/soe/`
Author's web page:	`http://www.reid-consulting-uk.ltd.uk/alastair/`

4.4.2 Win32 and Xlib Libraries

Report by: *Alastair Reid*
Project status: *Maintained, stable*
The Win32 library is a set of bindings to over 450 functions in the standard Windows C libraries.
Portability: *GHC, Hugs, Windows*
The Xlib library is a set of bindings to over 300 functions in the standard Xlib C library.
Portability: *GHC, Hugs, Linux, FreeBSD, Solaris, MacOS X*
Status: These libraries have been distributed for Hugs for some years and have worked with GHC for almost as long but they have suffered from second class status since they have always been treated as just one of the prerequisites for installing the HGL. We have recently moved them over into the hierarchical namespace and are almost ready to make a proper release.

4.4.3 HOpenGL – A Haskell Binding for OpenGL and GLUT

Report by: *Sven Panne*
Project status: *active, maintained*
The goal of this project is to provide a binding for the OpenGL rendering library which utilizes the special features of Haskell, like strong typing, type classes, modules, etc., but

is still in the spirit of the official API specification. This enables the easy use of the vast amount of existing literature and rendering techniques for OpenGL while retaining the advantages of Haskell over lower-level languages like C. Portability in spite of the diversity of Haskell systems and OpenGL versions is another goal.

HOpenGL includes the simple GLUT UI, which is good to get you started and for some small to medium-sized projects, but HOpenGL doesn't rival the GUI task force efforts in any way. Smooth interoperation with GUIs like gtk+hs on the other hand *is* a goal.

The latest release (version 1.04 on 21/01/03) marks the beginning of API transition: The GLUT binding has been rewritten completely, using only Haskell-98 features plus FFI and hierarchical modules, both widely available extensions. Furthermore, using quadrics and tessellators has been vastly simplified, offering a much more Haskell-like interface. The rest of HOpenGL is currently being rewritten, too, people interested in the latest development can check out the libraries part of GHC's fptools repository. A new release is planned for Q3 2003, offering:

- Full OpenGL 1.4 support
- Some ARB extensions
- An improved API, centered around OpenGL's notion of state variables
- Extensive hyperlinked online documentation
- No GreenCard dependency anymore

Further reading:

`http://www.haskell.org/mailman/listinfo/hopengl/`
`http://www.haskell.org/HOpenGL/`

4.4.4 FunGEn – A game engine for Haskell

Report by: *Andre W B Furtado*
Project status: *(still) being rebuilt*

The objective of the FunGEn project is to create a high-level game engine in and for Haskell. A game engine, roughly speaking, is a tool intended to help game programmers to develop games in a faster and automated way, avoiding them to worry about low-level implementation details. The main advantage of using a game engine is that, if it is built in a general and modular architecture, it can be used to develop many different (types of) games.

The first release of FunGEn (April/2002) consisted of a 2D platform-independent game engine, which implementation was based in HOpenGL (Haskell Open Graphics Library). It supported:

- Initialization, updating, removing, rendering and grouping routines for game objects; definition of a game background (or map), including texture-based maps and tile maps; loading and displaying 24-bit bitmap files;
- Reading and interpretating player's keyboard input; collision detection; time-based functions and pre-defined game actions; a few debugging and game performance evaluation facilities;
- Sound support (for windows platforms only... :-[ )

Some feedback indicated that the first version of FunGEn was not as "functional" as it was desired: some game issues were still being dealt through an imperative fashion. This way, the authors of this project decided to change the game engine philosophy: programmers should describe a game as a set of "specifications" rather than defining its behavior imperatively. The chosen alternative for accomplishing this task was to port the Clean Game Library (CGL) to Haskell, adding some FunGEn/(H)OpenGL specific features, such as mouse handling. Hence, the main idea is to rebuild FunGEn, in order to provide game programming mechanisms following the CGL concepts.

This rebuilding task, however, is taking much more time than initially planned. Some very complex (and still unexplained) environment setup problems froze the engine development for months. The authors of the FunGEn project concluded that the heterogeneous environment in which the software was being produced had become impracticable: Windows98 and Cygwin presented many compatibility problems regarding HOpenGL, GHC and FunGEn setup.

As the development environment has changed (WindowsXP is being used instead of Windows98), FunGEn finally had a chance to evolve again. Unfortunately, new problems are still challenging the project: some compatibility issues arising from the use of the most recent version of HOpenGL had to be solved and some difficulties regarding Haddock compatibility with Cygwin is demanding special attention.

Besides all problems, the authors of the project expect a new version to be released as soon as possible.

The final objective of FunGEn is to support both 2D and 3D environments, some game programming tools (such as map editors) and advanced game functionalities (such as multiplayer networking), although it is actually far away from that.

FunGEn is being maintained at the Informatics Center (Centro de Informatica) of Universidade Federal de Pernambuco (UFPE), by Andre W B Furtado (assisted by lecturer Andre Santos), and it's wide open for any implementation contributions. We would like to thank Mike Wiering, the creator of Clean Game Library.

Further reading:

`http://www.cin.ufpe.br/~haskell/fungen/`
`http://cleangl.sourceforge.net/` (Clean Game Library)

4.4.5 FunWorlds – Functional Programming and Virtual Worlds

Report by: *Claus Reinke*
Project status: *moved to HOpenGL, release delayed*

FunWorlds is an ongoing experiment to investigate language design issues at the borderlines between concurrent systems,

animated reactive 2&3d graphics, and functional programming. One of the aims is to get a suitable platform for expressing such things visually, preferably from Haskell, continuing from the start that functional reactive programming and especially Conal Elliott's Fran made in that direction.
With the reimplementation of FunWorlds on top of HOpenGL (earlier versions were based on VRML), the focus has shifted towards a redesign of some fundamental FRP concepts, towards a simpler operational semantics as a basis for uncomplicated implementations with more predictable performance characteristics. At the same time, I still want to be able to use Fran's high-level modeling approach.
The initial release has been delayed, simply because I haven't been able to allocate much time for this project. The stopping stone is not so much functionality (the first release will not have substantially more functionality than the snapshot presented at IFL'2002, so it will be pretty basic), but the need to write some form of introductory tutorial on the new DSEL design and how one might use it. Once we've got some more experience with the language basics, graphics functionality will be added on demand.
The good news is that trying to write the tutorial and examples has forced me to simplify the language still further (whenever something seems to require complicated explanations, I go back and try to simplify the design and implementation instead, maintaining expressiveness). If I don't manage to get back to this project soon to prepare a proper release, I'll probably just make snapshots available to interested parties.

Further reading:

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

4.5 Tool Frameworks

Instead of developing fixed tools, it is sometimes possible to generalize the code implementing the tool functionality into a library, so that the code can be reused for a family of tools.

4.5.1 Medina – Metrics for Haskell

Report by: *Chris Ryder*

The Medina library is a Haskell library for GHC that provides tools and abstractions with which to build software metrics for Haskell programs.
The library includes a parser and several abstract representations of the parse trees and some visualisation systems including pretty printers, HTML generation and callgraph browsing. The library has some integration with CVS to allow temporal operations such as measuring a metric value over time. This is linked with some simple visualisation mechanisms to allow exploring such temporal data. These visualisation systems will be expanded in the near future.
Currently we are working on some 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 have implemented several metrics using the library, which has given some valuable ideas for improvements.

Further reading:

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

4.6 XML and Web Programming

4.6.1 HaXml

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

HaXml provides many facilities for using XML from Haskell.

Recent news The latest release of the HaXml suite of libraries and tools is 1.09, just about to be published.

- Mostly a bugfix release, with some very minor changes to the API for reading and writing typed values to/from files, and a few minor configuration enhancements etc.
- HaXml is now available from CVS at `cvs.haskell.org`.

Future plans HaXml is in stable (bugfixes only) mode at the moment. Users regularly contribute improvements, which are usually adopted strightforwardly.

Further reading:

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

4.6.2 HXML

Report by: *Joe English*
Project status: *no developments, but actively maintained*

HXML is a non-validating XML parser written in Haskell. It is designed for space-efficiency, taking advantage of lazy evaluation to reduce memory requirements. HXML may be used as a drop-in replacement for the HaXml (section 4.6.1) parser in existing programs. HXML includes a module with functionality similar to HaXml's 'Combinator' module, but recast in an Arrow-based framework.
HXML also provides multiple representations for XML documents: a simple algebraic data type containing only the essentials (elements, attributes, and text), a tree representation which exposes most of the full XML Information Set, and a navigable tree representation supporting all of the principal XPath axes (ancestors, following-siblings, etc).
HXML has been tested with GHC 5.02, GHC 5.04, NHC 1.12, and most recent versions of Hugs. NHC 1.10 requires a patch.
HXML is basically in maintenance mode right now until I can find some spare time; support for XML Namespaces is next on the TODO list.

Further reading:

`http://www.flightlab.com/~joe/hxml/`

4.6.3 Haskell XML Toolbox

Report by: *Uwe Schmidt (uwe@fh-wedel.de)*
Project status: *second major release*

The Haskell XML Toolbox is a collection of tools for processing XML with Haskell. It is itself purely written in Haskell. The core component of the Haskell XML Toolbox is a validating XML-Parser that supports almost fully the Extensible Markup Language (XML) 1.0 (Second Edition).
The Haskell XML Toolbox bases on the ideas of HaXml and HXML, but introduces a more general approach for processing XML with Haskell. The Haskell XML Toolbox uses a generic data model for representing XML documents, including the DTD subset and the document subset, in Haskell. This data model makes it possible to use filter functions as a uniform design of XML processing applications. The whole XML parser including the validator parts was implemented using this design. Libraries with filters and combinators are provided for processing the generic data model.

Features:

- validating XML parser
- very liberal HTML parser
- XPath support
- full Unicode support
- support for XML namespaces
- uniform data modell for DTDs and XML content
- support of http: and file: protocol
- tested with W3C XML validation suite

Current Work:

- XSLT implementation
- better error reporting

Further reading:

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

4.6.4 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

- a monadic interface to generating HTML output
- type-safe compositional approach to specifying form elements; callback-style programming interface for forms
- automatic error detection
- complete interactive script in one program
- type-safe interfaces to state with different scopes: interaction, persistent client-side (cookie-style), persistent server-side
- integration with CSS yields compositional style descriptions
- on-the-fly generated graphics
- high-level interface to email generation

New/completed Items are:

- API for reading email
- Haddock-generated documentation
- support for hooking directly into Simon Marlow's Haskell Webserver
- support for generating forms that can be stored in files or sent via email or news
- simple support for frames
- beyond-Haskell98 interface: slightly simplifies programming of forms with more than one input
- incorporation of WASH/HTML, a typed interface for generating mostly valid HTML documents

Current work includes

- package-ifycation of WASH
- authentication interface
- new paper about WASH

Items still on the to do list

- preprocessor for translating markup in XML syntax into WASH/HTML
- database interface
- user manual

Further reading:

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

Chapter 5

Tools

5.1 Foreign Function Interface

5.1.1 C–>Haskell

Report by: *Manuel Chakravarty*
Project status: *beta release*
C–>Haskell is an interface generator that simplifies the development of Haskell bindings to C libraries. It has recently acquired support for conditional compilation, which facilitates supporting multiple versions of a C library within a single Haskell binding. Moreover, inline C declaration can be included into the binding, which is convenient if custom C code is needed for impedance matching between C and Haskell. The tool is currently at version 0.11.3 and it now has its own mailing list at <c2hs@haskell.org>. It has been stress tested in the development of the Gtk+HS GUI library.

Further reading:

Source and binary packages as well as a reference manual are available from `http://www.cse.unsw.edu.au/~chak/haskell/c2hs/`

5.1.2 GreenCard

Report by: *Alastair Reid*
Project status: *Maintained, stable*
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. Source and binary releases (Win32 and Linux) are available. The last release was 2.0.4 (August 2002). A release that provides access to and takes advantage of the new Foreign Function Interface libraries will be available soon.

Further reading:

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

5.1.3 Java VM Bridge

Report by: *Ashley Yakeley (<ashley@semantic.org>)*
Java VM 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.
It is intended to make writing "Java in Haskell" as straightforward as possible. To this end, each Java class is a separate type, and the argument lists of methods of automatically-generated interfaces to Java classes make use of subtype class relations to minimise explicit upward casting. Java exceptions are represented as Haskell monadic exceptions, and may be caught or thrown accordingly. Also, the two garbage collectors are integrated in such a way that cross-collector reference loops won't happen.
As a point of cleanliness and principle, it makes no use of "unsafe" Haskell calls (or pure function FFI). The layered design allows access to either lifted monads that keep track of context data (specifically, the JNIEnv pointer) and do all the work of preloading for you, or "IO"-based functions if you want to do all that yourself.

Current Status: A beta-quality 0.2 was released in April 2003 in source-code form. It compiles on Linux and Mac OS X with Sun's JVM, and should work with a number of others. Future plans include using the hierarchical module structure, separating out pure (non-FFI) Haskell into a separate package, and making the IO-based interface cleaner and friendlier.

Contact: Ashley Yakeley

Further reading:

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

5.2 Meta Programming

"Why write a program when you can write a program to write a program?" (author unknown).

5.2.1 Scanning, Parsing, and Analysis

See also constraint-based program analysis (section 3.3), the Parsec parser-combinator library (section 4.2.3) and the group of tools developed in Utrecht (section 6.4.5).

Alex

Report by: *Simon Marlow*

Alex version 1 **status:** *(not maintained)*

Alex is a lexical analyser generator for Haskell, similar to the tool lex for C. Alex takes a specification of a lexical syntax written in terms of regular expressions, and emits code in Haskell to parse that syntax. A lexical analyser generator is often used in conjunction with a parser generator (such as Happy) to build a complete parser.
Alex homepage (temporary):
`http://www.syntaxpolice.org/~ijones/alex/`

Alex version 2 **status:** *(in development)*

I've recently been developing a new version of Alex based on Chris Dornan's original code. The current status can be seen by browsing the "simonm-hackery-branch" branch of the CVS repository in `fptools/happy/alex`. The main aims are:

1. To make Alex work better with Happy. As a first step, this involves making the syntax more Happy-like (and indeed more lex-like). Ultimately having a combined grammar file and eliminating much of the tedious glue which is necessary between the parser and lexer would be desirable.
2. To improve the programmer experience. The aim is to abstract the lexer over (a) the input type and (b) the underlying monad, making Alex lexers more flexible in how they can be incorporated into a program.
3. To add support for Unicode.
4. To improve performance and size of the generated lexer code. Dramatic improvements have been made here, using traditional table compaction methods and some GHC extensions to improve performance of lexers when compiled with GHC.

The current status is that (1) is partly done (the syntax changes), (2) is partly done and (4) is done (but further table compaction could be done if it turns out to be necessary). (3) I have some ideas about.
Isaac Jones <ijones@syntaxpolice.org> has been writing the documentation for the next version of Alex.

Happy

Report by: *Simon Marlow*
Project status: *stable, maintained*

There have been no new releases of Happy since June of last year. Happy is still in constant use by GHC and other projects, and remains in maintenance mode.
There aren't any major developments on the horizon, although I have been putting some work into a new version of Alex which might have an impact on Happy; no details yet, but the aim is to make it easier to construct a complete parser using Alex and Happy together.

Further reading:

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

5.2.2 Haskell Transformations

Also see Strafunski's StrategyLib (section 4.2.1) for transformation support, and the Haskell Refactorer (section 5.3.2) for an application of transformations in program development.

MAG

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

MAG is a transformation tool for a small Haskell-like language which tries to make it easy to mechanise some complex program transformations with the help of user-supplied rewrite rules. Examples this has been tried with include cat-elimination and alpha-beta pruning.
Although not much time is being spent on it, it is still actively maintained and some work is being done on improving the higher-order matching algorithms that underpin its operation. We would of course be very happy to hear from anyone who is interested in using it (for example for teaching), or improving it. It was recently the basis for a chapter in *"The Fun of Programming"* (produced in honour of Richard Bird's 60th birthday).

Further reading:

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

HsOpt: Helium/LVM Optimization in Stratego

Report by: *Eelco Visser*

HsOpt is an optimizer for the Helium compiler implemented in the transformation language Stratego. Helium (section 2.5.2) is a subset of Haskell developed at Utrecht University (section 6.4.5). The optimizer works on the code produced by the Helium front-end, which is code for Daan Leijen's Lazy Virtual Machine (LVM). The goal of this project is to validate the paradigm of transformation strategies for the implementation of an optimizing compiler.
Alan van Dam has written the first version of the optimizer consisting of a basic simplifier in the style of the GHC. The main target of this simplifier has been the optimization of pattern matching code. The naive translation of pattern matching by the Helium front-end keeps it simple, but produces rather ugly code. Using a small set of transformation rules and an appropriate strategy the code can be reduced to more sane code, often similar to code that would be written by hand. This first simplification step produces an optimization of about 10%.
Plans for the near future are to include an inliner (currently only local let bindings are inlined, not global function definitions), and to incorporate the earlier work on deforestation.

Further reading:

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

5.3 Program Development

5.3.1 Tracing and Debugging

Report by: *Olaf Chitil and Bernie Pope*

By now 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 data structures at given program points. Hood and GHood are easy to use, because they are based on a small portable library. They have remained unchanged for over a year.

The Haskell tracing system **Hat** is based on the idea that a specially compiled Haskell program generates a trace file alongside its computation. This trace can be viewed with several tools in various ways: Hood-style observation of top-level functions; backwards exploration of a computation, starting from (part of) a faulty output or an error message. All tools inter-operate and use a similar command syntax. A tutorial explains how to generate traces, how to explore them, and how they help to debug Haskell programs. Hat can be used both with nhc98 and ghc. Hat can be used for Haskell 98 programs that use some language extensions.

On 26 March 2003 Hat 2.02 was released. That release added — besides numerous bugfixes and minor improvements — support for multi-parameter type classes and functional dependencies, provided your underlying compiler supports them, the hierarchical module namespaces, and includes a tracing version of (a subset of) the base package of standard hierarchical libraries.

For the future we plan to add a number of new tools for viewing the trace; in particular we want to resurrect a tool for algorithmic debugging of Haskell programs, which had been part of previous releases of Hat.

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 tree with nodes that correspond to function applications.

Currently buddha works with GHC version 5.04 or greater. No changes to the compiler are needed. There are no plans to port it to other Haskell implementations, though there are no significant reasons why this could not be done.

Version 0.5 is freely available as source. This version supports most of Haskell 98, however there are a few small items that are not supported. These are listed in the documentation. Future releases will include support for the missing features, and a much improved user interface. Version 0.6 should be available "real soon now" which has an automated front end.

Further reading:

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

5.3.2 Refactoring

Report by: *Claus Reinke*
Team: *Huiqing Li, Claus Reinke, Simon Thompson*

Refactorings are source-to-source program transformations which change program structure and organisation, but not program functionality. Documented in catalogues and supported by tools, refactoring provides the means to adapt and improve the design of existing code, and has thus enabled the trend towards modern agile software development processes. Refactoring has taken a prominent place in software development and maintenance, but most of this recent success has taken place in the OO and XP communities.

In our project *'Refactoring Functional Programs'*, we explore the prospects for refactoring functional programs, taking Haskell as a concrete case-study. Following the experiments we reported on last time, we have now established the meta-programming infrastructure we need by building on the Programatica (`http://www.cse.ogi.edu/PacSoft/projects/programatica/`) project's Haskell-in-Haskell frontend (providing parsing, static analysis and type analysis) and the Strafunski project's support for generic and strategic programming (see section 4.2.1). We are also acquiring the necessary expertise for integrating our prototype refactorer into Vim and Emacs, according to our survey the main Haskell program development environments.

Programatica's frontend supports nearly all of Haskell 98, and would be our frontend of choice, but it still hasn't been released, and there are still some question marks on its license and whether that would impede our own project. Strafunski has proven invaluable in avoiding unmanageable amounts of boilerplate code, enabling us to focus on the essence of program transformations when implementing refactorings over the complex Haskell grammar.

Built on top of all that infrastructure, work on our prototype refactoring tool is now progressing well, and the Haskell Refactorer already supports a handful of smaller-scale refactorings, like renaming, moving definitions, extracting and inlining definitions, and the like. A paper (*"Tool Support for Refactoring Functional Programs"*) describing some of the challenges in implementing such a tool is in preparation and the draft should be available from our project page within a couple of weeks. A first release of our Haskell Refactorer later this year should accompany the publication of that paper.

Further reading:

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

5.3.3 Testing

HUnit

Report by: *Dean Herington*

Project status: *maintained, update delayed*
Hunit is a unit testing framework for Haskell similar to JUnit for Java. With HUnit, a Haskell programmer can easily create tests, name them, group them into suites, and execute them, with the framework checking the results automatically. Test specification is concise, flexible, and convenient.
HUnit is free software that is written in Haskell 98 and runs on Haskell 98 systems. While written in Haskell 98, HUnit 1.0 is sensitive to the format of exception strings, which it inspects to judge the outcome of tests. The planned minor revision (HUnit 1.1), to adapt to recent GHC (5.04) and Hugs (Oct. 2002) versions, should get rid of this sensitivity, but has been delayed and will have to wait until after someone's Master's degree is finished.
The root cause for this problem seems to be the lack of extensibility in Haskell's exception mechanism. A tool such as HUnit shouldn't have to resort to tricks like the one involved here. It should be possible to cleanly define and use an "HUnit" kind of exception.

Further reading:

The software and documentation can be obtained at `http://hunit.sourceforge.net`.

5.3.4 Documentation

Haddock

Report by: *Simon Marlow*
Project status: *stable, maintained*
There have been no new releases of Haddock since July last year. A few bugfixes have been applied to the main source tree, along with a couple of minor improvements, so another release will be forthcoming soon.
Haddock is relatively stable, and I intend to keep maintaining it for the forseeable future. I don't have much time for wholesale improvements, although contributions are of course always welcome. 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`

Further reading:

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

Chapter 6

Applications, Groups, and Individuals

6.1 Non-Commercial Applications

This section lists applications developed in Haskell, be it in academia, in industry, or just for fun, which achieve some non-Haskell-related end.

6.1.1 HScheme

Report by: *Ashley Yakeley (<ashley@semantic.org>)*
Project status: *being rewritten*
HScheme will be a Scheme interpreter written in Haskell. There should be 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), or a fixed-point Scheme (that allows call-with-result), etc.

Current status: My first attempt was slow: programs spent a large amount of time looking up symbol bindings. You can play with the interpreter for this attempt on the web at `http://hscheme.sourceforge.net/interpret.html`.
I'm currently rewriting it to resolve programs' symbols before running them. I don't expect any releases in the next few months.

Further reading:

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

6.1.2 Haskell in Alchemy

Report by: *Alastair Reid*

Knit

Project status: *Active, maintained, no recent news*
Portability: *GHC (maybe Hugs, still), Linux, FreeBSD*
Knit is a component definition and linking language for systems programming based on the Unit component programming model. Knit lets you turn ordinary C code (e.g., bits of the Linux kernel) into components and link them together to build new programs. Since the freedom to do new things brings with it the freedom to make new errors, Knit provides a simple constraint system to catch component configuration errors. Knit also provides a cross-component inliner and schedules initialization and finalization of components.
Knit is released under a BSD-style license, is written in Haskell (and a little C) and includes a C parser and pretty-printer. A useful little utility included in the distribution is a tool for renaming symbols in ELF-format object files.
The current Knit release acts post-compilation: we compile C code as normal and then rename symbols in object files before linking. We are rewriting Knit to act pre-compilation: manipulating the source code before compilation. This will bring the much needed ability to import and export types from modules.

Flatten: Cross Module Inliner for C

Project status: *Active, maintained*
Portability: *GHC, Linux, FreeBSD*
Flatten can make existing, profiled, tuned and optimized C programs go 5-20% faster by performing performing inlining across module boundaries. More usefully, flatten lets you write C without making compromises to performance such as confusing interface with implementation (putting functions in header files) or writing overly large files.
Flatten works by parsing your C files, merging all the files into one, discards dead code, discards the user's guesses as to what functions should be inlined (this feature can be turned off), marking functions to be inlined, sorting function definitions to put definitions before uses and generating a single, giant C file for gcc to chew on. Flatten can cope with systems that contain assembly language, link with binary libraries, and other things that real code does. Flatten is an outgrowth from the Knit/Alchemy project which aims to bring some of the strengths of Haskell and Haskell tools to programmers on the street.
We are currently in the "release engineering" phase: more testing, add documentation, think of cooler name, commit to a license and package and aim to have a release out this summer.

Further reading:

`http://www.cs.utah.edu/flux/alchemy/`

6.1.3 Analysis Tools for Rosetta

Report by: *Perry Alexander*

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

Further reading:

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

6.1.4 Hircules, an IRC client

Report by: *Jens Petersen*

I have recently been working on an IRC client which I'm calling "Hircules" written in Haskell using gtk2hs for the GUI. It already works reasonably well and I expect to make an initial public release within the next week or so, and will then of course welcome contributions, bug reports, rfe's, etc. I took the code from lambdabot as the starting point, though it is already starting to diverge a bit. Currently it features tabs for channels and private conversions and retains lambdabot's bot features too. There is also an all channels tab and a raw IRC message tab for debugging and irc fanatics. I hope to add more features, like coloured nicks, utf-8 support, message alerting, etc, to follow.

Further reading:

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

6.2 Commercial Applications

6.2.1 Reid Consulting Ltd

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

Many companies are starting to allow their programmers to develop small prototypes in Haskell but few are willing to take a chance on using Haskell on a large project. The risks to these companies include lack of support for tools, lack of tutorials and training courses, gaps in the set of available libraries, and lack of 'gurus' to call on when things go wrong. Reid Consulting can meet those needs. Our background and continuing involvement in the development of Haskell tools and compilers (GreenCard, Hugs, GHC, etc.) and the Haskell language and library design (the Haskell report, the Standard libraries, the Hugs-GHC libraries, the Foreign Function Interface and the HGL Graphics Library) and our use of Haskell to develop large systems, provide the experience and the contacts needed for effective support of real projects.
Where acceptable to clients, we have a policy of releasing any fixes or developed code as OpenSource for use by the wider Haskell community.

Further reading:

`http://www.reid-consulting-uk.ltd.uk/`

6.2.2 Aetion Technologies LLC

Report: *Mark T.B. Carroll (<Mark.Carroll@Aetion.com>)*

Aetion Technologies LLC, a small defense contractor based in Columbus, Ohio, USA, is using Haskell in-house for most of its software development. Our software is largely a set of small- to medium-sized programs that are written to demonstrate a range of new capabilities to the Department of Defense.
Where helping the Haskell community coincides with our commercial aims, we manage to spare some time to do that. Activities, mainly led by Isaac Jones, include packaging for Debian GNU/Linux, writing documentation, working on the Wiki, answering people's Haskell questions, and reporting bugs.
Features that draw us to Haskell are: reusability; program correctness (supported by high-level features, a lack of side effects, and strong, static typing); cleanliness; and, multiple liberally-licensed compilers. We make use of multi-parameter typeclasses, and we have been using Haddock to help with our documentation. Probably, we see Haskell's main (fixable) weakness as being a shortage of dependable libraries that are well-maintained and that offer introductory and reference documentation for the working programmer.
Although the Haskell community may seem small and mostly academic, we believe that there is real potential for Haskell in industry, and that Haskell's future is sure enough to warrant the strategic decision of using it commercially. Hiring Haskell programmers, or people capable of quickly becoming such, does not seem to be a problem.
It is probable that, at some stage, a customer will require that a product be delivered in an "industry standard" language; it remains to be seen if it is then practical to continue research and development in Haskell, and to do ports to other languages when necessary.

6.2.3 Binary Parser

Report by: *Sengan Baring-Gould*
Project status: *no changes*

Sengan Baring-Gould <Sengan.Baring-Gould@nsc.com> at National Semiconductor is developing a binary parser which given a grammar is able to extract fields from values. This is used as part of an internal ICE (hardware debugger).
Binary parser provides the ability to reference by name values which may be composed of other values. It goes one step further in that the client program does not need to know where particular value is buried, only what its value is. Binary parser grammars are intended to enable non-programmers to

access fields of their registers, without requiring the ICE-developer to write explicit code to do so. For instance a technical writer could write a binary parser grammar for a device of which the ICE developer has never heard. Stress has been put on generality and simplicity, rather than efficiency. For instance binary parser allows multiple definitions, cyclic definitions, etc.

Binary Parser is implemented in Haskell whereas the current ICE is not (C++) – but the next generation will be. Currently communication is achieved using pipes so as to be compatible with both windows and unix (binary parser is used by 2 internal tools, one is unix one is windows).

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

6.3 Haskell in Education

6.3.1 Beseme Project

Report by: *Rex Page*

Studying connections between programming effectiveness and practice in reasoning about software.

Lecture notes comprise over 350 animated slides (all both PowerPoint and PDF formats). About two-thirds of the material 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. Several semesters worth of exams (finals and exams given during the term) are provided. The slides have matured through several offerings of the course.

The remaining third of the course discusses other standard topics in discrete mathematics, such as sets, functions, relations, trees, and counting. The web page 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). The web page provides a link through which instructors may gain access to the full website.

A statistical study of grades earned in a course for which discrete mathematics is a prerequisite shows a statistically significant difference between the grades of students who studied the Beseme materials and those of students who studied discrete mathematics in the traditional way. Beseme students earned higher programming grades, on the average, compared with students of comparable ability from the traditional group. Details of are available on the Beseme website.

Further reading:

`http://www.cs.ou.edu/~beseme/`

6.4 Research Groups

Many research groups have already been covered by their larger projects in other parts of this report, especially if they work almost exclusively on Haskell-related projects, but there are more groups out there who count some Haskell-related work among their interests. Unfortunately, we don't seem to reach some of them yet, so if you're reading this, please make sure that your group is represented in the next edition!

6.4.1 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 are centered on the UniForM workbench and the Common Algebraic Specification Language (CASL).

The **UniForM workbench** is an integration framework mainly geared towards tools for formal methods. It uses a simple, powerful and flexible notion of events to model all interactions between tools and users. In particular, the workbench provides HTk, an encapsulation of Tcl/Tk based on our event model (see section 4.3.2).

The workbench is actively used in the MMiSS project that aims to set up a multimedia internet-based adaptive educational system covering the whole subject of safe systems. The workbench currently contains over 80k lines of Haskell code (plus a few hundred lines of C), and compiles with the Glasgow Haskell Compiler, making use of many of its extensions, in particular concurrency.

We are also using GHC to develop tools, like parsers and static analysers, for languages from the **CASL** family, in particular CASL itself, HasCASL, and HetCASL.

Several parsers have been written using the combinator library Parsec (section 4.2.3). (Annotated) terms (from the ATerm Library) are used as a data exchange format and we use DrIFT (section 3.4) to derive instances for conversions. Documentation is generated using Haddock (section 5.3.4). (for ATerm see `http://haskell.org/libraries/`)

The CASL extension **HasCASL** combines specification and functional programming. The executable parts of a HasCASL specification are to be translated into Haskell (using the haskell-src package).

The language **HetCASL** is a combination of several specification languages used in formal methods, such as CSP, CASL, HasCASL, and Modal and Temporal Logic. We exploit Glasgow Haskell's multiparameter type classes, hierarchical name spaces, functional dependencies, existential and dynamic types.

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`

6.4.2 The Yale Haskell Group

Report by: *John Peterson*

The members of our group are *Paul Hudak, John Peterson, Henrik Nilsson, Antony Courtney, and Liwen Huang.*

The functional programming group at Yale is using Haskell and general functional language principals to design and implement domain-specific languages. We are particularly interested in domains that incorporate time flow. Examples of the domains that we have addressed include robotics, user interfaces, computer vision, and music. FRP was originally developed by Conal Elliott as part of the Fran animation system. It has three basic ideas: continuous- and discrete-time signals, functions from signals to signals, and switching. FRP is particularly useful in hybrid systems: applications that have both continuous time and discrete time aspects.

Yampa is the culmination of our efforts to support domain-specific embedded languages using FRP. Yampa, packaged with a robot simulator, is currently in release (see section 4.2.5)

John Peterson is working on DSL's for educational use (see section 2.5.3). *Antony Courtney* is working on yet another graphics library for Haskell to provide capabilities similar to the Java 2-D graphics library. This is in support of functional GUIs constructed using FRP. *Henrik Nilsson* is using FRP principles in a hybrid modelling language. He is interested in non-causal modelling in which a system is described by a system of signal relations rather than functions.

Liwen Huang is working on a language which describes the motion of humanoid robots. *Paul Hudak* works a little bit on each of the above topics, and also on the Haskore computer music library. Most recently he's developed an algebraic theory of polymorphic temporal media.

Further reading:

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

6.4.3 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 (section 3.2.2) in a Mosix-Cluster environment. At this time, after correcting some small bugs in the run time system, we have the current version of the GUM run-time system for parallel Haskell operating, and, in order to make the system more compatible with Mosix, we are currently removing the PVM dependency. A preliminary version of PVM-free GUM is currently being tested.

In coming months we expect to apply parallel Haskell to research on pseudorandom number generation (the shuffled nested Weyl generator of Holian, Percus, Warnock and Whitlock, Phys. Rev. E., <50>, 1607 (1994)) and image processing for extremely-high resolution medical-image visualization.

Further reading:

`http://www.sci.brooklyn.cuny.edu/~metis`

6.4.4 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.

- We are looking at using our port of Haskell (see section 2.5.1) for embedded DSLs to build handheld applications.
- *Kate Krastev* is investigating specialisation of the nhc98 runtime with a view to code compaction.
- *Phuong Tri* is working on program proving for Haskell using Isabelle.
- 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.

Further reading:

Our web page is currently being re-developed. In the meantime, please contact us via email to <plrg@ics.mq.edu.au>.

6.4.5 Functional Programming at Utrecht University

Report by: *Doaitse Swierstra*

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

Parser Combinators *(Doaitse Swierstra, Arthur Baars, Rui Guerra)*
The current version of the parser combinators constructs an online result, in the sense that parts of the result can be accessed even when parsing has not yet finished. This is especially useful when parsing and processing large files of similar information. Furthermore error messages are displayed while parsing (using unsafePerformIO). The underlying mechanism for achieving this is relatively costly, although parsing speed is not much slower than that of parsers generated off line using Frown or Happy (section 5.2.1).
Furthermore we added combinators that construct parsers that reorder the recognized elements (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 retain the original input in such a way that error messages can be easily added to it. We also added a combinator that 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.

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

Improving Type Errors *(Bastiaan Heeren, Jurriaan Hage, Doaitse Swierstra)*
See the description of the constraint based type inferencer in section 3.3.2.

Generic Haskell *(Johan Jeuring, Andres Loh, Dave Clark, Doaitse Swierstra)*
See the description in section 3.4.

The attribute grammar system AG *(Arthur Baars, Doaitse Swierstra)*
The AG system offers Haskell programmers a seamless way to integrate the "attribute grammar" programming style with Haskell. This allows you to separate aspects of a program in more powerfull way than laziness and functions alone offer. The AG system has for example been used extensively in the implementation of Helium compiler to specify aspects like pretty printing, static analysis, and other tranformations.
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, that may run faster. The system is used in the course on Implementation of Programming Languages.
We are now 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 unfortuantely incorrect use leads to pages of error messages. We hope that following the techniques explained in `http://www.cs.uu.nl/people/arthurb/dynamic.html` may help to solve the problem.

Type Checker for Extended Haskell *(Atze Dijkstra, Doaitse Swierstra)*
As a companion to Mark Jones' "Typing Haskell in Haskell" we are constructing a type inferencer for full (extended) Haskell. Some of its features are a consistent way of handling existential and polymorphic types, and the use of polymorphic kinds (if you want to know what they are good for read the "Typing Dynamic Typing" paper presented at the ICFP2002, `http://www.cs.uu.nl/people/arthurb/dynamic.html`). We are currently rounding of this construction with the less interesting, but more laborious parts of full Haskell. We plan to use this material later this year in a course on "Type Systems".

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.

Proxima *(Martijn Schrage, Johan Jeuring, Lambert Meertens, Doaitse Swierstra)*
Proxima is a generic graphical structure editor with support for free editing (ie. normal typing instead of selecting transformations from menus) and computations over the data structure. The system has a layered architecture, which is described and implemented using a library of architecture combinators. For the presentation of the document data structure, the graphical presentation combinator library Xprez has been developed. The user interface will be implemented using the upcoming wxHaskell GUI library (section 4.3.5).
One of the intended applications of Proxima is an editor/IDE for the language Helium. It will support editable pretty printed code in which types and error messages can be shown. Sources can be edited by normal typing (also for changing layout) as well as by performing structural edit commands.
A prototype of Proxima is expected to be ready in the second half of 2003.

Syntax Macros *(Arthur Baars, Doaitse Swierstra)*
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. Most of the necessary glueing code is automatically generated. The syntax macros make it possible to extend the context free grammar of a language on a per program basis. Examples of constructs that no longer have to be part

of the standard language, but could have been defined using our macro mechanism are the do-notation, and the notation for list comprehensions. An open question, on which we work, is how to provide feedback to the user in terms of his original program. The current version is available at: `http://www.cs.uu.nl/people/arthurb/macros.html`

6.4.6 Functional Programming at the University of Kent

Report by: *Claus Reinke*

Here at what is now called the University of Kent, the functional programming interest group currently includes about a dozen people pursuing research interests in functional programming. Haskell is a major focus of teaching and research, although we also look at other languages (such as Erlang `http://www.erlang.org`).

Our members pursue a variety of Haskell-related projects, many of which are reported in other sections of this report. *Keith Hanna* is continuing his work on visual interactive programming with Vital (see section 2.5.4). *Axel Simon* develops the Gtk2hs binding to version 2.2 of the Gtk GUI library (section 4.3.7). *Chris Ryder* is now evaluating his Metrics and Visualization library Medina (section 4.5.1) through some case studies.

Huiqing Li, *Simon Thompson* and *Claus Reinke* are making good progress building a prototype Haskell Refactorer (section 5.3.2), and Claus Reinke is also making slow progress and would like to be able to spend much more time (...) on his project combining functional programming and virtual worlds (section 4.4.5).

Stefan Kahrs works on the expressiveness of programming languages, e.g. what one can do with certain language features in Haskell. In this vein is his work on red black trees `http://www.cs.kent.ac.uk/~smk/redblack/rb.html` and showing, via a Haskell implementation, the relative completeness of PCF `http://www.cs.kent.ac.uk/~smk/PCF/Untyped_PCF.html` .

Further reading:

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

6.4.7 Programming Languages & Systems at UNSW

Report by: *Manuel Chakravarty*

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

Further reading:

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

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

Report by: *Phil Trinder*

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.

Members: *Abyd Al Zain, Andre Rauber Du Bois, Hans-Wolfgang Loidl, Robert Pointon, Greg Michaelson, Phil Trinder, Jan Henry Nystrom, Mustafa Khalifa Asawd, Norman Scaife, Chunxu Liu, Graeme McHale*

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 analyse 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/`

6.5 Individual Haskellers

"What are you using Haskell for?" – the implementation mailing lists are full of people sending in bug reports and feature suggestions, stretching the implementations to their limits. Judging from the "reduced" examples sent in to demonstrate problems, there must be quite a few Haskell applications out there that haven't been announced anywhere (probably because Haskell is "just" the tool, not the focus of those projects).
If you're one of those serious Haskell users, why not write a sentence or two about your application? We'd be particularly interested in your experience with the existing tools (e.g., that all-time-favourite: how difficult was it to tune the resource usage to your needs, after you got your application working? Which tools/libraries where useful to you? What is missing?).

Tom Pledger <Tom.Pledger@peace.com> writes:
Since 2001 my work has mainly been the design and implementation a business data processing language. It is lazy, functional, and has a lot in common with discrete Functional Reactive Programming. I tried and failed to implement it as a Haskell library, before committing to a separate language.
A prototype of the runtime virtual machine, implemented in Haskell, passed some initial tests in April 2002. We expect to get a text based prototype working in May 2003, complete with its database interface and interpreter front end. We have plans for a graphical user interface, comprising an Integrated Development Environment and a runtime debugger.
I take care not to ask for related free advice on the Haskell mailing lists, because my employer expects to own and profit from what I produce.

Graham Klyne (`http://www.ninebynine.org/`, `http://www.ninebynine.net/`) writes:
My primary interest is in Semantic Web (`http://www.w3.org/2001/sw/`) and RDF technologies, and I am a participant in the W3C RDFcore working group (`http://www.w3.org/2001/sw/RDFCore/`). I'm a developer rather than a researcher, and am currently working toward using RDF in network configuration applications. I aim to develop inference rules to map general network policy descriptions (in RDF) to device-specific configuration files and/or instructions. I see the Web in general, and the Semantic Web in particular, as a natural territory for application of functional programming techniques. The availability of open, high quality language implementations, and a vibrant user community are important considerations in choosing Haskell.
I aim to use Haskell to explore simple inference techniques for RDF data, motivated in part by limitations I have encountered when using simple off-the-shelf RDF inference tools. I find it particularly appealing that Haskell has the full power of a general purpose programming language, but supports programming styles that can be matched very closely to formal specifications, thus providing extra validation for Internet/Web protocol specifications. So far, in Haskell, I have released a URI processing package and test cases (`http://www.ninebynine.org/Software/Intro.html#URIsInHaskell`) with a URI parser based closely on a proposed revision of the URI specification, and am currently working on an RDF graph API, graph-isomorphism tester, Notation3 parser (`http://www.w3.org/DesignIssues/Notation3.html`) and Notation3 formatter (mostly working, but not yet released). With the recent availability of XML parsers with namespace support, I'd also like to tackle a full RDF/XML parser based on the new proposed RDF syntax specification (`http://www.w3.org/TR/rdf-syntax-grammar/`).
Once I have some basic RDF processing tools in place, I plan to apply them to network access control and configuration applications.

Dean Herington <herington@cs.unc.edu> writes:
I'm presently completing a Master's thesis for which I built a small domain-specific language (embedded in Haskell, of course) to coordinate the execution of cooperating but independent application programs. Participating programs and data files are described formally. A compound execution–termed a federation–is expressed in Haskell, extended with operations for large-grain program description and coordination. The declarative expression of a federation in terms of data flow among the component programs captures synchronization requirements implicitly and can exploit the inherent concurrency automatically. Haskell compilation, notably its rigorous type checking, ensures the consistency of the federation. I applied the coordination framework to one large problem, the federation of several existing Fortran programs that simulate environmental processes in the Neuse River estuary of North Carolina.

Oleg Kiselyov <oleg@pobox.com> continues to extend his collection of Haskell programming miscellanea, exploring algorithms and programming techniques, with extensively commented example code `http://pobox.com/~oleg/ftp/Haskell/index.html`: One recent addition is a page that discusses several approaches to number-parameterized types, i.e., datatypes that depend on unary or decimal numbers. `http://pobox.com/~oleg/ftp/Haskell/number-parameterized-types.html`
Examples include arrays parameterized by their length (like an array type in Pascal), bitfields of a statically known size and Z_n types for modular groups or rings with a statically known modulus. An attempt to add, for example, two bitvectors (arrays, modular integers) of different sizes results in a compiler error with a clear error message. Often the numerical parameter is an unary (Church) numeral. We will also show an implementation of number parameterized types with decimal numerals. The latter are far easier to use. We also

demonstrate arithmetic on (decimal) number parameterized types, which lets us statically typecheck operations such as array concatenation.

We should stress that the basic number-parameterized types can be implemented entirely in Haskell98. Advanced operations such as type arithmetic require Haskell extension to multi-parameter classes with functional dependencies.

The pure-functional λ-calculator project has been developed further and has a new web page:
`http://pobox.com/~oleg/ftp/Computation/lambda-calc.html`

The project implements a domain-specific language of the pure untyped λ-calculus embedded into Haskell. The new project page describes several applications of the calculator:

- λ-calculus of negative numbers and division (the code has been ported to the notation of the Haskell-based calculator)
- P-numerals: arithmetically more convenient and efficient numerals than Church numerals. P-numerals are a functional equivalent of a list data structure.
- A solution to a bluff combinator problem in λ-calculus

Mike Thomas <mthomas@gil.com.au> writes:

I'm working on Haskell bindings to the Japi (C wrapped Java GUI), Grass (geographic information system), MPICH (parallel processing by message passing) and Proj (map projection) libraries to allow me to read, write, display and process GRASS mapsets and to do some geochemical modelling, hopefully with the ability to distribute large map computations with MPICH.

Windows GHC is my compiler of choice and until recently I relied heavily on the Parsec and ObjectIO libraries, each of which is an excellent development tool.

However, the demise of the GTK ObjectIO library port led me to drop it as the GUI component of the project. Instead, I released a Haskell binding to a substantial subset of the Japi GUI library. Although that library is not as slick as ObjectIO, the Haskell binding is easily ported to other platforms and compilers and is also easily maintained by myself without depending on the trials and tribulations of third party libraries or on other programmers' coding assistance. To add practical weight to this aspect of the binding I recently commenced work make the binding work with NHC98. (On the side, I also made a complete binding for GNU Common Lisp which is available from the Savannah CVS repository.)

I have also changed direction on my interface to GRASS, preferring now to port libgis to the MinGW32 Windows C compiler rather to rewrite substantial chunks of functionality in Haskell. Although the Haskellisation process was an interesting experience, there is now little left for me to gain by that duplication, and much to lose in terms of extra work as GRASS itself evolves through time. In general I think that unnecessary duplication in the open source community can lead to dissipation of part time programmer-years which might otherwise be better spent.

The Japi Haskell binding is publicly available in the libraries subdirectory of the GHC fptools CVS tree. The remainder of this work is not available on the web as it is relatively incomplete and I have not decided on a release model.

Further reading:

on GRASS:	`http://grass.itc.it`
on MPICH:	`http://www-unix.mcs.anl.gov/mpi/mpich/`
and Japi:	`http://www.japi.de/`
