Haskell Communities and Activities Report

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

– second edition –

May 8, 2002

Claus Reinke (editor), University of Kent at Canterbury, UK
Manuel Chakravarty, University of New South Wales, Australia
Olaf Chitil, The University of York, UK
Antony Courtney, Yale University, USA
Sigbjorn Finne, Galois Connections, USA
Andre W B Furtado, Federal University of Pernambuco, Brazil
Andy Gill, Galois Connections Inc., USA
Dean Herington, University of North Carolina at Chapel Hill, USA
Johan Jeuring, Utrecht University, The Netherlands
Daan Leijen, Utrecht University, The Netherlands
Rita Loogen and Steffen Priebe, University of Marburg, Germany
Jan-Willem Maessen, Massachusetts Institute of Technology, USA
Simon Marlow, Microsoft Research Cambridge, UK
Henrik Nilsson, Yale University, USA
Johan Nordlander, Chalmers University, Sweden
Ross Paterson, City University London, UK
Simon Peyton Jones, Microsoft Research Cambridge, UK
Robert Pointon, Heriot-Watt University, UK
Bernie Pope, University of Melbourne, Australia
Chris Reade, Kingston University, UK
Alastair Reid, Reid Consulting (UK) Ltd., UK
Satnam Singh, Xilinx Inc, USA
Martin Sulzmann, University of Melbourne, Australia
Wolfgang Thaller, Graz, Austria
Peter Thiemann, University of Freiburg, Germany
Phil Trinder, Heriot Watt University, UK
Malcolm Wallace, The University of York, UK
Ashley Yakeley, Seattle WA, USA

Preface

Another 200 or so emails later, the editor and the many contributors are happy to present the second edition of the Haskell Communities and Activities Report (I'm still looking for that magical instance declaration `instance Haskeller p => DeepSeq p where ..`).

The idea behind these reports is simple: twice a year (currently, in April and October), a call goes out to the main Haskell mailing list, inviting all Haskellers to contribute brief summaries of their area of work, be it language design, implementation, type system extensions, standardisation of GUI APIs, applications of Haskell, or whatever. The summaries introduce the area of work, the major achievements over the previous six months, the current hot topics, and the plans for the next six months. They also provide links to further information.

So, every six months (currently, in May and November), all Haskellers should get a bird's-eye view of the Haskell community as a whole, and pointers to more in-depth information. This makes it a lot easier to try and keep up to date with what the various specialist communities and projects are working on, without having to follow all of the dozens of relevant mailing lists. It should also make it possible for individual Haskellers to find the communities they are most interested in, and to join their efforts. Projects and individuals currently working separately on related topics might feel encouraged to coordinate their work and cooperate.

Those interested in specific tools or resources can find them via the Haskell home page `http://www.haskell.org`, but the Activities Report will tell you which new entries have become available over the last six months, which of the existing entries are still actively being worked on, and what the recent and planned developments for these are. Even just knowing that a certain tool is still being maintained, and who felt responsible for it when the latest edition of this report came out, can be important information if you want to base your next project on it.

Last, but not least, the reports also give non-Haskellers (or not-yet Haskellers;-) an opportunity to get an overview of what is currently going on in "Haskell Land", helping them to decide whether to go deeper in, and where. Welcome to our new visitors – be sure to visit the Haskell home page and the two main mailing lists as well (see section 1.1). If you are already a Haskeller and find yourself wanting to explain what it is all about to someone you know (your boss, perhaps?-), you might find it helpful to pass on the latest edition of this report.

To the specialist communities, these bi-annual reports offer an occasion for distributing discussion documents to the main community, asking for feedback and for contributions. The Revised Haskell 98 Report is stabilizing (section 1.2), and be sure to check the documents describing the new Foreign Function Interface (section 3.1) and the new Hierarchical Module Namespace (section 3.2).

My greatest concern for this second edition was whether there would be much new to report after just six months. There are a few projects which will provide major updates only in every second edition (the reports affected by this are explicitly marked as unchanged). Other projects that reported declining activity in the last edition have been dropped in this one, as these reports focus on recent activities only. Some entries have shrunk to little more than valuable pings, confirming that the project/software is still actively maintained, and giving a contact.

As it turns out, all of this has been more than offset not only by a lot of activity reported in the other projects, but by several completely new entries and a growing chapter on Haskell applications. So far, it seems that the six month interval for our reports can be sustained easily by new developments. And so I'm sure you'll find this edition at least as interesting a read as the first one, thanks to the many volunteers whose contributions make up the heart of this report, and thanks to many more whose work gives us something interesting to report on.

In spite of the wealth of topics covered, this second edition still does not cover all current work on or with Haskell. Little chance of that – Sydney and Melbourne recently reported that each of them introduces more than 1000 new first-year students to Haskell every year. But I hope that still more Haskellers will contribute summaries of their favourite Haskell topics in the future. Please put October 2002 into your diaries now! Someone will be asking for your contributions to the November 2002 edition of this report then!-)

Claus Reinke,
University of Kent at Canterbury, UK

Contents

Chapter 1

General

1.1 Haskell Central - WWW and Mailing Lists

`http://www.haskell.org`
Haskell's central information resource, has the language and standard library definitions, links to Haskell implementations, libraries, tools, books, tutorials, people's home pages, communities, projects, news, conferences & workshops, a wiki, question & answers, applications, educational material, job adverts, Haskell humour, and even merchandise. Be sure to visit, there may be parts you haven't noticed.
haskell.org also hosts most of the Haskell-related mailing lists and CVS repositories (15 mailing lists at a recent count, plus about another dozen of CVS-related lists). While the overall structure of the web site has been relatively stable for some time now, the maintainers John Peterson and Olaf Chitil are aiming to keep the contents in each part up to date.
Most Haskell-related information is reachable from haskell.org, and anything that isn't, should be. Do not just wait for John or Olaf to pick URLs and infos from lengthy messages in long-running threads on the Haskell lists: *send new or updated entries (category + link + short description) directly to John or Olaf.*
And please, could everyone take the release of this Communities Report as an occasion for going through the information on haskell.org relating to our own interests and send in updates, where appropriate? As its says on haskell.org: "This web site is a service to the Haskell community. The site is maintained by John Peterson and Olaf Chitil. Suggestions, comments and new contributions are always welcome. If you wish to add your project, compiler, paper, class, or anything else to this site please contact us."

Further reading:

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

1.2 Revised Haskell 98 Report

The revision has been in bug-fix-only mode for a while now, and Simon Peyton Jones's declared plan as the editor has been to freeze the report after a month with no bug reports. Mischievously, though, people always seem to keep back a bug or two, sending them in for the bug of the month (no, there is no such competition;-), always just in time to reset the freezing criterion.
It's probably a good sign that Haskellers are finally testing, and asking for clarification of, all the odd corners in the Haskell Report. But as the contents have been fixed for quite a while now, and are already being used as the reference point for implementations, it might be time to move the Revised Report into place over the next couple of months, treating any further bugs there.

Further reading:

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

1.3 Tips, Tricks, Tours and Tutorials

It seems that some Haskellers have documented their own hard-won experience to help others. They have been working on web pages, short papers, tours, and tutorials touching on introductory examples of monads&co, giving guided tours and explanations of prelude, libraries & syntax, or tips about programming and resource tuning, even explaining the internals of GHC (scary;), or interpreting Hugs error messages.
With this new section, we'll try to enhance the visibility of such valuable resources, but ultimately, all these things should be linked from the Haskell bookshelf (have another look, it is not limited to books:):
`http://www.haskell.org/bookshelf/`
One such resource is "A Tour of the Haskell Prelude", originally a paper by Bernie Pope, intended as a guide to the functions, operators, and classes of the Haskell 98 Prelude, now HTML-ised and updated by Arjan van IJzendoorn:
`http://www.cs.uu.nl/~afie/haskell/tourofprelude.html`
Arjan has recently added "The Tour of the Haskell Syntax", an overview of the Haskell syntax, intended as a teaching aid for use alongside Haskell textbooks:
`http://www.cs.uu.nl/~afie/haskell/tourofsyntax.html`
Miloslav Nic <nicmila@systinet.com> has been working on a Haskell Reference, based on Haskell 98 Report and Haskell 98 Libraries Report:
`http://zvon.org/other/haskell/Outputglobal/`

He writes: "The current version is of beta-release quality. There are unfinished sections in it and some functions are missing, but I hope that it is already useful.

In the final version there should be examples of use for every construct, links to relevant materials discussing usage patterns, theoretical matters, source codes, ... There is a long way to go, but I can be patient (see Zvon RFC repository for a proof.) If you have some relevant examples and/or links, please, send them to me."

Those who encounter the oracle of Hugs error messages for the first time will have noticed that they tend to tell the truth, but challenge you to find it!-) In those circumstances, Simon Thompson's collection of "Some common Hugs error messages" might be helpful.

`http://www.cs.ukc.ac.uk/people/staff/sjt/craft2e/errors/allErrors.html`

He writes: "Working out why Hugs gives you a particular error message can be tricky. This page pulls together a collection of error messages and the code that produced them; the entry for the error message you have provoked can hopefully help you to diagnose your particular problem"

Oleg <oleg@pobox.com> suggests his collection of Haskell programming miscellanea: "This web designs several monads (including a Monte Carlo monad to operate 'fuzzy numbers' of arbitrary distribution). Also included an illustration of an ST monad for emulating a CPU with predicated instructions. The page also contains the description, correctness proof and an optimal pure functional implementation of the perfect shuffle algorithm."

`http://pobox.com/~oleg/ftp/Haskell/misc.html`

He also gives an introductory example of monadic programming – in Scheme! He juxtaposes Haskell code for a particular state monad with the corresponding Scheme code:

`http://pobox.com/~oleg/ftp/Scheme/monad-in-Scheme.html`

Not for beginners, but certainly necessary, is Manuel Chakravarty's GHC Commentary: "The Glasgow Haskell Compiler (GHC) Commentary aims to explain the magic behind GHC. It is an evolving resource that describes the structure and to some extent the implementation details of various subsystems of GHC and is mainly directed at people who like to tinker with GHC. As the Commentary is relatively young and GHC rather huge, only part of the system is covered so far. The master copy of the Commentary is located in GHC's CVS repository, so that all developers can contribute their wizardly insights. The CVS version is mirrored at the following web site, where it is updated daily:"

`http://www.cse.unsw.edu.au/~chak/haskell/ghc/comm/`

And even if you're not a compiler hacker, you'll sooner or later run into the problem of controlling the resource usage of your Haskell programs (analysing resource usage is better covered, and supported by various useful profiling tools). Amanda Clare has documented tools and techniques that have been useful to her while using Haskell for data mining in bioinformatics data:

`http://users.aber.ac.uk/ajc99/stricthaskell.html`

1.4 Haskell-related Publications

Last time, we reported on what looks to be a very interesting special issue on Haskell (`http://www.cs.nott.ac.uk/~gmh/jfp.html`, expected publication sometime in 2002) in The Journal of Functional Programming. Thanks to Graham Hutton, guest editor for that special issue, we had the titles, authors, and abstracts for the six papers that will appear in it. We'll try to turn this into a permanent section pointing to recent Haskell-related publications (books, conference proceedings, special issues in journals, PhD theses, etc.), with brief abstracts. In future editions, this will be coordinated with Jim Bender's "Online Bibliography of Haskell Research" (`http://haskell.readscheme.org`).

This time, we have a fun demonstration of program derivation in the context of the Countdown game, Keith Wansbrough's thesis on simple polymorphic usage analysis, and a paper on "Fine Control of Demand", which, in case you didn't guess from the title, includes "a calculational, dynamic semantics of a large subset of Haskell". Combined with the work we reported on last time, this indicates further progress in closing the embarassing gaps in Haskell's formal basis, a very welcome development.

"**The Countdown Problem**", *Graham Hutton*; to appear as a Functional Pearl in the Journal of Functional Programming, 2002.

This paper develops a Haskell program to solve the numbers game from Countdown, a popular quiz show on British television. The aim wasn't to produce solutions as fast as possible (although in absolute terms the final program actually performs rather well), but to show how the program itself could be developed in systematic way in conjunction with a proof of its correctness. It's the kind of example that can be covered as part of a Haskell course, and the powerpoint slides that I produced for my Haskell course in Nottingham are freely available. The paper, source code, and slides can be obtained on the web from:

`http://www.cs.nott.ac.uk/~gmh/bib.html#countdown`

"**Fine Control of Demand**", *William Harrison, Tim Sheard and James Hook*; to be published this July at MPC2002 (Mathematics of Program Construction) held in Dagstuhl.

Just how does Haskell differ from the lazy lambda calculus? We answer this question by introducing a calculational, dynamic semantics for a large subset of Haskell that exposes the interaction of its strict features with its default laziness. In the semantics, features perturbing Haskell's standard lazy evaluation order are specified computationally (i.e., monadically) rather than as pure values (i.e., functions, scalars, etc.).

`http://www.cse.ogi.edu/~wlh/`

Keith Wansbrough (`http://www.cl.cam.ac.uk/~kw217/`) has recently completed his PhD, "**Simple Polymorphic Usage Analysis**" (`http://www.cl.cam.ac.uk/~kw217/research/phd/index.html`). He developed a type-based analysis that discovers the "usage" of each thunk in a Haskell program – either at most once, or possibly many times. If

a thunk is known to be used at most once, it need not be updated with its value after evaluation, and it may be inlined without wasting work. A number of other optimisations are also enabled by this information.
The approximating analysis developed for the purpose, simple polymorphism, is of independent interest and should be applicable to other problems. The analysis was implemented in GHC, and measurements showed a moderate performance benefit; more work remains to be done in making use of the information provided by the analysis. It has not yet been decided whether the analysis will be integrated with released versions of GHC.

If you are interested in further Haskell-related research publications, be sure to have a look at Jim Bender's "Online Bibliography of Haskell Research" (`http://haskell.readscheme.org`). He writes: "My aim has been to gather together links to research publications (technical reports, theses, conference publications, etc.)–and to actively maintain this collection. Though in a sense there is overlap with the "Bookshelf" at Haskell.org, my aim is different. The "Bookshelf" is, in effect, a best of collection, with a focus on being tutorial in nature (at least currently). My focus is more narrow–to concentrate on research only–but the coverage (of research publications) is also more extensive."
And if you still haven't come across the Haskell bookshelf, you'll find it at `http://www.haskell.org/bookshelf/`. It lists textbooks, papers (especially of tutorial nature), proceedings of the "Advanced Functional Programming" summer and spring schools, as well as reference material, often created in the context of Haskell courses (see also our tips&tricks section 1.3).

Chapter 2

Implementations

2.1 The Glasgow Haskell Compiler

Report by: *Simon Peyton-Jones*

The Team

Simon Peyton Jones, Simon Marlow (with particular help recently from Sigbjorn Finne, Koen Claessen, Wolfgang Thaller)

Current status

We released GHC 5.02.3 in early April. 5.02 is a nice, stable compiler and we intend to treat it as our stable baseline for some time to come.
Meanwhile, there have been quite a lot of developments on the `HEAD`, leading to two 'snapshot' releases of GHC 5.03. (A snapshot release comes with a health warning; it's an alpha-quality thing.) Our plan is to produce a full 5.04 (or perhaps 6.00!) release in May.

New stuff in 5.03/6.0

- The big thing in 5.03 are the new hierarchical libraries (see section 3.2) GHC now fully supports the new library system.
- Going along with the new libraries is a Haskell documentation tool, Haddock, written by Simon M. It provides rather spiffy HTML-browsable documentation by processing the Haskell source code in a somewhat intelligent way. See section 5.3.3.
- Heap profiling has been beefed up significantly, with support for retainer profiling and biographical profiling (lag/drag/void) a la Nhc98. We used the new tools on GHC itself, and fixed several space leaks (the changes were backported into the 5.02.2 release).
- The GHC commentary keeps on growing. If you've not had a look, you might like to: `http://www.cse.unsw.edu.au/~chak/haskell/ghc/comm/`
- Type system things:

  1. Arbitrary-rank polymorphism is now fully implemented. Mark Shields and Simon PJ are writing a paper. This means that you can have functions with truly bizarre types like:

     ```
     f :: ((forall a. Ord a => a->a)->Int)->Int
     ```

     provided you supply enough type signatures (not many). Incidentally, GHC has supported rather general type synonyms for some time, so you can abbreviate like this:

     ```
     type T = forall a. Ord a => a -> a
     f :: (T -> Int) -> Int
     ```

  2. Linear implicit parameters are an experimental feature. They feature implicit "splitting" of an implicit parameter so you can distribute (say) a unique supply, or a random-number supply around your program using implicit parameters. Whether this is a good thing or not is a moot point. Documented in the user manual.
  3. Generalised 'deriving' for newtypes. You can now say

     ```
     newtype T = MkT RepT deriving( MyClass )
     ```

     Provided that `RepT` is an instance of `MyClass` (which need not be a built-in class), the instance of `MyClass RepT` will be 'lifted' to `T`. John Hughes suggested the idea. It's a bit more general than it looks here, as the user manual shows.
  4. Optional explicit kind annotations at the binding site of type variables. E.g.

     ```
     data T (k :: *->*) = T (k Int)
     ```

     This can be useful when there isn't enough context from the type declaration to describe the kind you want.

- There is now an almost-complete port of GHC to MacOS X, thanks mostly to the work of Wolfgang Thaller <wolfgang.thaller@gmx.net>.
- Mark and Simon wrote a paper about scoped type variables as implemented in GHC. `http://research.microsoft.com/~simonpj/papers/scoped-tyvars/`
- Interface files (Foo.hi) are now in binary format. (Use `ghc --show-iface Foo.hi` to see it in readable format.) This is faster to print and parse, and easier to extend. There will be a re-usable library for the `Binary` class; but we need to discuss with folk what it should look like. Any takers?
- Koen Claessen wrote a parser monad to support the `Read` class. It should provide parsers that are much

smaller (in code size) and much faster to execute than the existing implementation. The underlying library is useful in its own right, and is exposed as `Text.ParserCombinators.ReadP`. The lexical analyser is exposed as `Text.Read.Lex`

- Sigbjorn has implemented support for Haskell-calls-C-calls-Haskell, which involves quite a bit of cunning. It's documented in the GHC commentary. [Not quite complete.]
- The new Foreign Function Interface is almost fully implemented (thanks to Manuel C for doing most of the work). `http://www.cse.unsw.edu.au/~chak/haskell/ffi/` The omissions relate only to the C API for calling Haskell from C.
- Generic classes are working again.

Remaining on the agenda

- GHC.NET
- Non-blocking I/O for Windows
- Meta Haskell

Further reading:

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

2.2 Hugs

Report by: *Sigbjorn Finne*

Team / status

The Hugs98 interpreter is now maintained by Sigbjorn Finne and Jeffrey Lewis, both of Galois Connections. The previous maintainer, Johan Nordlander, has moved on to pastures new, but still helps out whenever possible.
Since the last community report (Nov '01), a new major release of Hugs98 was released in December; a release consolidating fixes and additions made during 2001. It has proven to be a good, stable one.

Future plans

Apart from inching Hugs98 closer to the Haskell98 standard, the next big thing for Hugs is to switch over to using the new Haskell hierarchical library. When this will happen is dependent on the amount of free time available to the maintainers over the next couple of months, but our best estimate of when this will be done is before the summer (of 2002 :-) is up.

Further reading:

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

2.3 nhc98

Report by: *Malcolm Wallace*

Current Status

The most recent version of nhc98 is 1.12, released in March 2002. It continues to receive bugfixes and minor improvements, primarily in response to reports and requests on the mailing list. The current part-time maintainers are Malcolm Wallace and Olaf Chitil, but as with any open-source project, all of you are encouraged to send patches, add extensions, or even adopt components if you wish. (Many thanks to the contributors who have already submitted improvements.)

Highlights

- The Hat tracing system continues to improve in usability, and now in addition to the original version that is fully integrated with nhc98, a new portable version that works with ghc is also included in the distribution. Eventually, Hat will be fully separated into a compiler-independent tracing/debugging system.
- Coming in the next release of nhc98 is an 8% improvement in the runtime of compiled programs - thanks to a one-line(!) patch from Thomas Nordin. Thomas is also responsible for various other time and space performance improvements in the compiler itself.

Lowlights

- Although nhc98 remains the most portable Haskell'98 compiler on Unix-like machines, it has recently become more difficult to build nhc98 on a Windows/Cygwin machine if you already have ghc installed. We hope to fix at least some of these issues over the summer.

Future Plans

- The next release (1.14) will probably occur in early Summer 2002.
- We also plan to move our CVS repository to cvs.haskell.org, where more people can potentially have immediate access for development.
- After that, we still want to add the same set of utility libraries to nhc98 that are already distributed with both Hugs and ghc, once they are converted fully to the new hierarchical naming scheme.
- The new syntax for the standard FFI will probably be adopted around the same time as the hierarchical libraries.
- Beyond that, plans for new features are in your hands! Hack them yourself, or suggest them on the nhc-users mailing list.

Further reading:

`http://www.cs.york.ac.uk/fp/nhc98/`

2.4 Eager Haskell

Report by: *Jan-Willem Maessen*
Project status: *currently slowed down by thesis*
Compiles arbitrary Haskell programs, but happens to run them eagerly using resource-bounded execution. There should be no difference in observed program behavior—every Haskell program is a valid Eager Haskell program (except that our compiler doesn't yet cover all of Haskell 98—we're missing qualified names and field names).
Except for the missing bits of language and libraries, this is a real honest-to-goodness Haskell implementation. It's even easy to hack.

Goals:

Make it easy to write efficient programs (eg tail-recursive loops) in Haskell. Explore the efficiency tradeoffs between eager and lazy execution. Extract parallelism from ordinary haskell programs without annotating them.

Status:

Eager Haskell is still chugging along; those interested in the Hacker's Release should drop me a line at <jmaessen@mit.edu>. My thesis is basically complete, and I'm defending May 8, so I'm a bit preoccupied at the moment.

People:

me. Thus the stealthy pace.

Further reading:

`http://csg.lcs.mit.edu/~earwig/eager-haskell.html`

Chapter 3

Language Extensions

3.1 Foreign Function Interface

Report by: *Manuel Chakravarty*
Project status: *Version 1.0 almost stable*
Release Candidate 4 of the Haskell 98 FFI Addendum has just been circulated for public review. The changes between different versions of the release candidates are minor and, for all practical purposes, the addendum can be regarded as stable. The current version of the addendum is available from `http://www.cse.unsw.edu.au/~chak/haskell/ffi/`
GHC now supports the FFI extension as defined in the addendum, in addition to the pre-standard syntax for backward compatibility. Other systems still need to be revised to conform to the addendum.

Further reading:

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

3.2 Hierarchical Module Namespace

Report by: *Simon Marlow*
There hasn't been any significant activity since the previous report. However, I'm expecting things to ramp up in two ways now: firstly the next release of GHC will have the hierarchical libraries in their full glory, and secondly I'd like to start documenting what we have using Haddock. Then I'm hoping the other implementations will come on board (Hugs is planning to ship with the full set of hierarchical libraries in its next release, I'm not sure about nhc98).

Further reading:

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

3.3 Non-sequential Programming

3.3.1 Concurrent Haskell

Report by: *Simon Marlow*
Project status: *no changes since last time*
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/set/sec-concurrent.html`

3.3.2 GpH – Glasgow Parallel Haskell

Report by: *Phil Trinder*
Recent work covers language, system and applications aspects, and consistently emphasises the architecture independence (cf. `http://www.cee.hw.ac.uk/~dsg/gph/arch-indep.html`) of our approach. The latest version of GpH (GUM-4.06) is available for RedHat-based Linux machines (binary snapshot `ftp://ftp.cee.hw.ac.uk/pub/gph/gum-4.06-snap-i386-unknown-linux.tar`; installation instructions `ftp://ftp.cee.hw.ac.uk/pub/gph/README.GUM`). A version for Sun shared-memory machines is available on request <gph@cee.hw.ac.uk>. More information on research projects, group members and publications is available from GpH `http://www.cee.hw.ac.uk/~dsg/gph/papers/`

Language We have produced a truly parallel implementation of a referentially transparent bottom-avoiding choice operator (`http://www.cee.hw.ac.uk/~dsg/gph/papers/drafts/flops-submitted.ps.gz`) and used it to explore a new class of parallel algorithms in GpH, namely branch-and-bound. It reveals an interesting relationship between non-strict and speculative parallel evaluation.

System In order to improve the architecture independent performance of GpH we have added new features to its implementation (GUM). The load balancing (`http://www.cee.hw.ac.uk/~dsg/gph/papers/drafts/sfp01-gum.ps.gz`) in GUM has been made more flexible by implementing low- and high-watermarks on the spark pools, which represent potential parallelism. Thread migration is being implemented as a technique of avoiding gross load imbalance in applications with a small amount of parallelism. For a better control of *data locality* in GpH programs we are currently exploring

language constructs with explicit placement parameters as well as abstractions over these basic constructs. We have improved the distributed shared memory performance (`http://www.cee.hw.ac.uk/~dsg/gph/papers/ps/dsm02.ps.gz`) of GUM, in particular global address management and the graph packing, enabling the user to optimise the parallel execution for execution time or heap space.

An implementation of a time and space static analysis is nearing completion. Although the current analysis is for a strict language, the intention is to use the result of the analysis to select appropriate computations for parallel evaluation.

Applications We have produced detailed comparisons of three parallel functional languages : GpH (`http://www.cee.hw.ac.uk/~dsg/gph/`), Eden (`http://www.mathematik.uni-marburg.de/~loogen/eden.html`), PMLS (`http://www.cee.hw.ac.uk/Research/funct_prog.html`), discussing language and implementation differences (`http://www.cee.hw.ac.uk/~dsg/gph/papers/drafts/hosc-submitted.ps.gz`). Detailed performance results of several parallel programs on a Beowulf cluster are given. A survey of parallel and distributed Haskells (`http://www.cee.hw.ac.uk/~dsg/gph/papers/ps/jfp01.ps.gz`) will also soon appear in the JFP special issue.

We are investigating the architecture independence of GpH by developing a significant application (genetic alignment) for a variety of parallel architectures: a Beowulf cluster, a Sun-Server SMP. We have also published careful measurements of the Naira parallel Haskell compiler.

Further reading:

`http://www.cee.hw.ac.uk/~dsg/gph/`

3.3.3 GdH – Glasgow Distributed Haskell

Report by: *Robert Pointon*

The following projects are ongoing:

1. We are investigating alternative distributed computation and communication strategies. Non-strict languages allow a range of strategies in a distributed system: e.g. the client may issue request messages serially, or as a group of requests, and the requested values may be computed sequentially or in parallel, and by client or server.
2. We are investigating the design and implementation issues of introducing mobile computations into our family of languages.
3. We have won a £180K EPSRC research project to investigate the value of high level programming techniques for developing distributed telecommunications software in Erlang and GdH. The project is in collaboration with Motorola UK Research Labs, and work will start in May. More information on this project is available here (Telecoms Project Homepage `http://www.cee.hw.ac.uk/~dsg/telecoms/`).
4. We have recently been investigating the use of GdH to prototype parallel extensions for GpH (`http://www.cee.hw.ac.uk/~dsg/gph/`).

Further reading:

Papers describing these projects are available from:

1. GdH Homepage `http://www.cee.hw.ac.uk/~dsg/gdh/`
2. Phil Trinder's Homepage `http://www.cee.hw.ac.uk/~trinder/publications.html`

3.3.4 Eden

Report by: *Rita Loogen and Steffen Priebe*

Project status: *no changes, work ongoing*

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. Differences include stepping back from a global heap to a set of local heaps to reduce system message traffic and avoid global garbage collection. The current (freely available) implementation is based on GHC 3.xx. An Eden implementation based on GHC 5.xx will be available in the near future.

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.

Current and future topics include program analysis, skeletal programming, and polytypic extensions.

Further reading:

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

3.4 Type System/Program Analysis

3.4.1 A General Type Class Framework based on Constraint Handling Rules

Report by: *Martin Sulzmann*

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. We are currently trying to integrate a type-debugging tool into Chameleon.

Further reading:

CHR-type systems general: `http://www.cs.mu.oz.au/~sulzmann/chr/`
Chameleon: `http://www.cs.mu.oz.au/~sulzmann/chameleon/`

3.4.2 Program Analysis for Haskell

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

Our goal is to develop a generic constraint-based program analysis framework for Haskell. Haskell improves programmer productivity, but compilers require complex program analyses to make programs run efficiently.
We have designed and implemented a binding-time, strictness and exception analysis for Haskell and incorporated both analyses into the GHC compiler. The analysis deals with all features of Haskell such as polymorphic programs and structured data.
The team: Kevin Glynn, Harald Sondergaard, Peter Stuckey, Martin Sulzmann

Further reading:

`http://www.cs.mu.oz.au/~sulzmann/mupag/`

3.5 Generic Programming

Report by: *Johan Jeuring*
Project status: *only small changes in topics&goals*

Software development often consists of designing a datatype, 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. 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.

3.5.1 Preprocessors

DrIFT (`http://www.cs.york.ac.uk/fp/DrIFT/`) is a preprocessor which generates instances of generic functions. It is used in Strafunski (`http://www.cs.vu.nl/Strafunski/`) to generate a framework for generic programming on terms.

3.5.2 Languages

PolyP (`http://www.cs.chalmers.se/~patrikj/poly/`) is an extension of a subset of Haskell in which generic functions can be defined and type checked. Polyp allows the definition of polytypic functions on a limited set of datatypes. Hinze has shown how to overcome some of the limitations of Polyp by extending Haskell with a construct for defining type-indexed functions with kind-indexed types. Generic Haskell (`http://www.generic-haskell.org/`) is based on Hinze's ideas. Also GHC has an extension that uses Hinze's idea to add derivable type classes to Haskell.

Current Hot Topics: Generic Haskell: XML tools as generic programs, an implementation of type checking and inferencing, adding views and data types as fixed-points, different styles of generic definitions.

Major Goals: Extend Generic Haskell with features that simplify the construction of XML tools, and several other generic programming problems, such as programming on terms. Next release of Generic Haskell: somewhere this summer. Strafunski: first-class generic functions.

Further reading:

`http://www.cs.york.ac.uk/fp/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.

3.6 Syntactic Sugar

3.6.1 Arrow Notation

Report by: *Ross Paterson*

A preprocessor for arrow notation was reported at ICFP'01. This has now been packaged by the Yale FRP group, who are using it in a new arrow-ized version of FRP, which should appear soon. The preprocessor itself is fairly stable, but I'm still taking requests, and am very keen to hear from any users. A library of arrow transformers is under development.
`http://www.haskell.org/arrows/`

Chapter 4

Libraries

4.1 Graphical User Interfaces

4.1.1 GUI Library API Task Force

Report by: *Manuel Chakravarty*
Project status: *idling...*
The main goal is the development of a GUI library **API** for Haskell that is portable across Haskell systems and operating/windowing systems. While several Haskell GUI libraries are now available (see below), no progress has been made on defining a common minimal API.

Further reading:

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

4.1.2 Object I/O for Haskell

Report by: *Krasimir Andreev*
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 but the library is done keeping in mind its portability. The Linux version based on GTK+ for Haskell is being developed. The aim is to create a highly portable GUI library. In this way the programs will be translated to different platforms without rewriting. The second requirement to the library is to give the programs a native look and feel for the target platform. The main difference between Object I/O and TclHaskell, FranTk and some other, is that Object I/O uses a native interface (Win32 API for Windows and GTK+ for Linux) instead of a scriptable interface (Tcl/Tk). This is more difficult to implement but is more effective.
The library uses nonstandard type system extensions: explicit universal quantification and existentially quantified data constructors, which aren't compliant with Haskell98 standard, but are a part of what we call Haskell-2-pre specification. The current implementation works only with GHC-5.02 or higher compatibles and is a part of hslibs collection. There aren't any plans to port the library to other compilers (NHC and/or Hugs). The package is distributed together with the port of original examples contributed with Clean Object I/O. These examples help the customers understand how to work with the library and how to understand the differences between the implementation for Haskell and Clean (for these who have experience with Clean). There is also a draft of Object I/O quick reference.
The port supports all features except features related to printing. Maybe I will implement printing in the future. Currently I redesign the library to implement modern hierarchical module names. Porting GUI applications from Clean to Haskell is not trivial but relatively easy task. In many cases the translation is just syntactical

Further reading:

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

4.1.3 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 the toolkit on which the Gnome desktop is based. GTK+ is a fully-fledged modern widget set and all its basic and some of its advanced functionality is already available from Haskell. The current binding is to GTK+ 1.2, but it will be extended to also support the new GTK+ 2.0 API in the next couple of months.
Gtk+HS is an API binding; that is, it remains close to the original C API, which implies a very stateful way of programming in the IO monad. More functional layers on top of the basic binding are under investigation. Gtk+HS includes support for two non-standard extra widgets: GtkGLArea supports OpenGL-based 2D and 3D graphics in GTK+ interfaces and GtkEmbedMoz facilitates embedding the rendering engine (Gecko) of the Mozilla web browser into Haskell programs. Moreover, the visual GUI builder Glade (`http://glade.gnome.org/`) can be used to design interfaces for Gtk+HS.

Further reading:

`http://www.cse.unsw.edu.au/~chak/haskell/gtk/`

[Axel Simon has recently released an alternative Haskell binding to Gtk2: `http://gtk2hs.sourceforge.net/` (ed.)]

4.2 Graphics

4.2.1 HGL Graphics Library

Report by: *Alastair Reid*
Project status: *Maintained, stable*
The HGL gives the programmer access to the most interesting parts of the Win32 and X11 library without exposing the programmer to the pain and anguish usually associated with using these interfaces. The library is distributed as open source and is suitable for use in teaching and in applications. The library currently supports:

- Filled and unfilled 2-dimensional objects (text, lines, polygons, ellipses).
- Bitmaps (Win32 version only, for now).
- Control over text alignment, fonts, color.
- Simple input events (keyboard, mouse, window resize) to support reactivity.
- Timers and double-buffering to support simple animation.
- Use of concurrency to avoid the usual inversion of the code associated with event-loop programming.
- Multiple windows may be handled at one time.

To keep the library simple and portable, the library makes no attempt to support:

- User interface widgets (menus, toolbars, dialog boxes, etc.)
- Palette manipulation and other advanced features.
- Many kinds of input event.

Status: The library works on both Win32 and X11 under Hugs and (unsupported) GHC. The API is stable and the library is used throughout Paul Hudak's 'School of Expression' textbook (`http://haskell.org/soe/`). The last release was 2.0.4 in December 2001.

Further reading:

HGL web page: `http://haskell.org/graphics/`
School of Expression web page: `http://haskell.org/soe/`
Author's web page: `http://www.cs.utah.edu/~reid/`

4.2.2 Haven, a Functional Vector Graphics Library

Report by: *Antony Courtney*
Project status: *Active, maintained, (relatively) stable*
Haven is a library for vector graphics in Haskell. Haven supports a number of features, including bezier curves, high-quality fonts, anti-aliased rendering, alpha-blending (transparency), constructive-area geometry and more. Haven presents a purely functional API, but is implemented using the Java2D renderer.

Further reading:

For more information on Haven, including examples and download instructions, please visit the haven web page at:
`http://www.haskell.org/haven/`

4.2.3 HOpenGL – OpenGL Haskell Binding

Last time, Sven Panne reported that his HOpenGL had been used by a handful users over the last two years and that it seemed to have gained some momentum recently. Well, that certainly rings true now, as we see the first two HOpenGL-based projects in this edition. As usual for Haskell applications, it is not always easy to find a clear boundary between libraries, embedded domain-specific languages, and stand-alone applications. You'll find FunGEn, a functional game engine, here in the libraries section (section 4.2.4), and VOP, a POV-Ray scene previewer, in the applications part (section 6.1.1). HOpenGL itself has recently (21/04/2002) seen another compatibility release: "Apart from some minor fixes to make it work with GHC >= 5.x and some strange GLU versions, it includes support for 3D textures. The installation procedure has been vastly improved, too.".

Further reading:

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

4.2.4 FunGEn - Functional Game Engine

Report by: *Andre W B Furtado*
Project status: *new project*
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 a game programmer to develop games in a faster and automated way, avoiding him 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) consists of a 2D platform-independent game engine, whose implementation is based in HOpenGL (Haskell Open Graphics Library). It supports:

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

The final objective of FunGEn is to support both 2D and 3D enviroments, 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 of the Federal University of Pernambuco, by Andre W B Furtado (assisted by lecturer Andre Santos), and it's wide open for any implementation contributions.

Further reading:

`http://www.cin.ufpe.br/~haskell/fungen/`
`http://www.cin.ufpe.br/~haskell/hopengl/`
`http://www.haskell.org/HOpenGL/`

4.3 Web Programming

4.3.1 WASH/CGI – Web Authoring System for Haskell

Report by: *Peter Thiemann*
Project status: *new project*
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

Current work includes

- incorporation of WASH/HTML, a typed interface for generating mostly valid HTML documents
- preprocessor for translating markup in XML syntax into WASH/HTML
- database interface
- authentication
- 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*
The FFI binding generator C–>Haskell has recently been extended with support for semi-automated argument and result marshalling, which significantly reduces the amount of code the author of a binding has to write. A binary release of the 0.10 series is forthcoming.

Further reading:

`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 2.0.3 (November 2001).

Further reading:

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

5.1.3 GCJNI

Report by: *Antony Courtney*
Project status: *Active, maintained, (relatively) stable*
GCJNI is a library that allows Haskell to invoke Java code via the Java Native Interface (JNI). The implementation uses GreenCard to make the JNI (a C language interface) available to Haskell. GCJNI includes a few convenient features, such as:

- Integration of the Haskell and Java garbage collectors, so that Java objects are garbage collected when they are no longer accessible from Haskell.
- Type class based overloading, which makes it easy to pass common types (like Int, Float and String) to or from Java code.
- A tool (GenBindings) which uses Java reflection on a set of compiled Java classes to generate a Haskell module with a simple, high-level, type-safe interface to the underlying Java code.

GCJNI has been successfully tested using both hugs98 and ghc under both Linux and Windows. The distribution includes a unified Makefile system and detailed release notes that makes it very easy to configure, compile and install for any supported combination of Haskell implementation and platform.

Further reading:

More information (including pointers to the relevant distributions) is available from the GCJNI web page at:
`http://www.haskell.org/gcjni/`

5.1.4 Java VM Bridge

Report by: *Ashley Yakeley*
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 0.1 was released in December 2001, for x86 Unix only. Release 0.2 will also be available for Windows and MacOS X, just as soon as stable 5.04 GHC is available on those platforms (unless I get tired waiting).

Contact: Ashley Yakeley <ashley@semantic.org>

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).
Even in a language where functions are first-class citizens, you sometimes want to write programs at a meta level, be it to get that extra leverage in productivity, to test some ideas for language extensions, for debugging/instrumenting your code, or for analyses and transformations. Unfortunately, generic tool support for this kind of tasks has been somewhat lacking, so that Haskell meta-programmers currently have to implement their tools almost from scratch (Drift, HAT, Sugar for Arrows, Haddock, labelled fields before they became part of the language, ...). In this section, we hope to document any progress being made in this area.

5.2.1 Haskell Frontends

Report by: *Bernie Pope*

Parsing/Printing Haskell Source

HsSource A full parser for Haskell 98 source, based on the Happy parser generator (see section 5.4.1), is maintained in the GHC CVS repository:
`http://cvs.haskell.org/cgi-bin/cvsweb.cgi/fptools/hslibs/hssource/`
The project is currently active and the parser is kept up to date with the Haskell 98 Language Report. The basic components of the project are a lexer, a parser, a pretty printer and an abstract syntax representation. The parser requires Happy.
Infix applications are parsed without reference to precedence levels of operators. This means that the abstract syntax representation may need restructuring after parsing is finished. Currently the project does not support this restructuring. The difficulty is that infix rules may be imported from other modules, and there is no way to know these infix rules when parsing a single module, without (partially) parsing other modules also. This is not a limitation of the code, but rather a limitation imposed by the definition of Haskell 98.
The parser is designed to process a whole module at a time, which makes parsing only fragments of code difficult or impossible. However, there is no such limitation on the pretty printer. The pretty printer can output code in layout sensitive and layout insensitive modes. Error messages are very minimal, and would require improvement for tasks that need to provide detailed feedback to the user.
This project is well suited to whole module analysis and whole module transformation, particularly because of the convenient abstract syntax representation.

Type inference/checking

Bernie Pope (`http://www.cs.mu.oz.au/~bjpop/`) and others at Melbourne University are working on a type checking tool for Haskell 98. The parsing is done by HsSource (see above). The typing algorithm is based on Mark Jones' "Typing Haskell in Haskell" (`http://www.cse.ogi.edu/~mpj/thih/`).
This is work in progress. Currently the program can type a single module program which imports the Prelude. Multi-module programs are not yet supported, however, most of the infrastructure is ready to do so.
One of the main aims of the project is to provide detailed information about the static properties of a Haskell module including:

- identifier binding locations, binding methods, and dependency relationships,
- the type class hierarchy,
- kinds, and
- types for top and let/where bound identifiers and data constructors

A release for the current version of the project is due within a couple of weeks of this version of the Haskell Communities Report. We hope that other interested parties would like to collaborate, or contribute some work on the code to support more of the Haskell language, so that it may be a useful tool for the Haskell community.

[At least two other related efforts are known (see the November 2001 edition of this report, section 1.3, for further details), but we have not been able to get hold of more recent information on these. (ed.)]

A note on why the front-ends of existing compilers do not meet the needs of the Meta Programming community

One would think that instead of writing another parser and type checker for Haskell you could simply extract the front end of one of the existing compilers/interpreters. There are really two main problems with this:

1. The front-ends of the compilers tend to be designed with the rest of the compiler in mind, thus making it difficult to select only the parsing and typing code out from them.

2. The compilers support their own specific set of extensions to the language which effect some aspects of the parsing and typing code. From this position it is harder to make a parser and type checker follow the Report correctly.

Having a reference parser and type checker which is independent of the compilers might also help us to reason about the static properties of Haskell 98 in a non-compiler-specific manner.

5.2.2 Haskell Preprocessors

DrIFT

DrIFT is Noel Winstanley's utility for deriving instances of Haskell classes. If you feel you are writing almost the same class instances again and again (observe? record field update?), this is one place to look for help. The tool formerly known as something else may face yet another name change (jDrIFT?), but more importantly, it seems to have acquired a new maintainer. John Meacham <john@foo.net> has not only started to add new features, he is also working on bringing together the various useful patches other DrIFT users have been making:

"It appears that there is popular support for a maintained version of DrIFT somewhere. I would be most willing to take over as project maintainer. Judging from the emails I received, it appears that many people have written small patches to DrIFT but with no where to send them, they just kept them to themselves. Since there already is a version in the fptools CVS, it would make sense to use that as the main repository for development, I guess a good plan of action would be for me to create a main web page (...) to distribute it from and then bring the cvs tree up to date with all my changes + the others people are submitting and then develop from there."

Further reading:

`http://homer.netmar.com/~john/computer/haskell/DrIFT/`

5.3 Program Development

5.3.1 Tracing and Debugging

Report by: *Olaf Chitil*

There exist several tools with rather different approaches to tracing Haskell programs for the purposes of debugging and program comprehension.

Hood, the portable library for observing data structures at given program points, has remained unchanged in the last half year. John Meacham's recent version of DrIft supports the generation of instances of the Hood class `Observable` for user defined data types, thus making Hood nearly as easy to use with any Haskell compiler as with Hugs. The latter directly supports a variant of Hood. GHood, the graphical back end for Hood which can animate observations, remains unchanged as well.

Freja, the algorithmic debugger for a subset of Haskell has not been developed further. The algorithmic debugger Buddha is not yet available, but it is making progress. Its developer Bernie Pope recently released low-level libraries for accessing the heap of ghc from Haskell.

In the middle of March, version 1.12 of Hat, the Haskell tracing (and debugging) system, appeared. Hat can now be used not only with nhc98 but also with ghc, that is, Hat transforms a Haskell 98 program into another Haskell program which can be compiled with ghc. The compiled program generates a trace file alongside its computation. With several new or improved tools the trace can be viewed in various ways: algorithmic debugging a la Freja; Hood-style observation of top-level functions; stack-trace on program abortion; 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 new tutorial introduction to Hat shows how to generate traces, how to explore them, and how they help to debug Haskell programs. The next improved version of Hat will appear end of May.

Further reading:

`http://www.haskell.org/libraries/#tracing`

5.3.2 Testing

HUnit

Report by: *Dean Herington*
Project status: *new tool*

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

QuickCheck

No recent developments have been reported for this tool, but as the Testing section is new here, Koen Claessen and John Hughes' QuickCheck `http://www.cs.chalmers.se/~rjmh/QuickCheck/` should be mentioned.

QuickCheck is a tool for testing Haskell programs automatically. The programmer provides a specification of the program, in the form of properties which functions should satisfy, and QuickCheck then tests that the properties hold in a large number of randomly generated cases. Specifications are expressed in Haskell, using combinators defined in the QuickCheck library. QuickCheck provides combinators to define properties, observe the distribution of test data, and define test data generators.

5.3.3 Documentation

Haddock

Report by: *Simon Marlow*
Project status: *new tool*
Haddock is a new tool for generating interface documentation from Haskell source. It takes a collection of plain Haskell source modules, optionally containing documentation annotations in the form of special comments, and generates an HTML or DocBook interface document. The documentation is fully hyperlinked and contains an index.
Because Haddock parses the source and contains a partial implementation of the Haskell module system, it can generate an accurate description of the API exported by a module. It also handles re-exporting of entities correctly.
There's still lots to do, and comments & suggestions are welcome.
Haddock's home page is here:
`http://www.haskell.org/haddock/`

5.4 Scanning and Parsing

5.4.1 Happy

Report by: *Simon Marlow*
Nothing significant since the previous report.
Paul Callaghan has sent me code to implement GLR (generalised LR) parsing written by a student of his (Ben Medlock). GLR parsing generates *all* the parses for a particular input, rather than selecting one of the possible parses when the grammar is ambiguous. The multiple parse trees are represented in a compact way. GLR parsing will hopefully be incorporated in the next release of Happy.
`http://www.haskell.org/happy/`

5.4.2 Parsec

Report by: *Daan Leijen*
Parsec is a monadic combinator library for parsing. Allthough combinator parsing is well known in literature, most libraries are only implemented for personal use or for small research examples. Parsec is designed from scratch as an "industrial-strength" parser library. It is safe, documented, has extensive libraries and good error messages. It is also fast, doing thousands of lines per second on todays machines, which might make it an acceptable alternative to bottom-up parser generators like Happy.
Parsec is currently part of the GHC libraries and part of the CVS repository: `anonymous@cvs.haskell.org:/home/cvs/root`, in `fptools/hslibs/text/parsec`
Pre-packaged releases and documentation can be found on the Parsec home page: `http://www.cs.uu.nl/people/daan/parsec.html`
The current library is quite complete but (in the next 6 months?) we plan to add:

- combinators for parsing Haskell "layout"
- a full Haskell parser
- error-recovery combinators
- more documentation, especially about the proper use of the "try" combinator

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 VOP – Vision of Persistence

Report by: *Wolfgang Thaller*
Project status: *new project*
VOP ("Vision of Persistence") is a freeware program that reads POV-Ray scene description files and displays them using OpenGL. POV-Ray (`http://www.povray.org`) is a freeware ray-tracing program that can create very realistic images and animations, but takes a lot of time to do so. VOP is intended to speed up the trial-and-error cycle by providing a fast low-quality preview. VOP consists of almost 10000 lines of Haskell code, plus a few lines of C++ which could all be reimplemented in Haskell.
Source code and binaries for MacOS X, Linux/x86 and Windows are available from `http://www.kfunigraz.ac.at/imawww/thaller/wolfgang/vop-intro.html`

6.1.2 Knit

Report by: *Alastair Reid*
Project status: *Active, maintained, semi-stable*
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.
Or, from a functional programmer's perspective, Knit is a 2nd-order lazy functional programming language (i.e., a linker) with an obscure variant of Haskell's type class mechanism to detect cycles and which uses a dependency analysis to determine a feasible evaluation order for the initialization routines. :-)
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.
Current work aims to extend error checking into the real-time domain, to automate generation of components, and to turn Knit into an architecture description language (ADL) instead of just a module interconnection language (MIL).

Further reading:

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

6.2 Commercial Applications

6.2.1 Lava at Xilinx

Report by: *Satnam Singh*
Lava has been used at Xilinx to design several high performance circuits that were difficult or almost impossible to produce using conventional techniques. Examples include signal processing and image filters, sorting networks, arithmetic networks (e.g. high speed pipelined adder trees) and distributed serial arithmetic implementations of arithmetic functions and filter blocks. Lava is distinguished from conventional languages by its ability to express the layout of circuits in a tractable and flexible manner. Lava also greatly eases the rapid design of new circuit cores by allowing designers to define new "glue" to compose circuits (circuit combinators implemented as higher order functions).
The Xilinx Lava implementation can generate VHDL, Verilog, SystemC [soon] as well as implementation netlist formats like EDIF which can be decorated with precise layout information. These outputs can be fed into a conventional tool flow to produce real circuits for download onto Xilinx's FPGAs.
Xilinx Lava is currently being updated to provide support for interfacing to external tools like SAT-solvers and model checkers. This will allow us to perform circuit transformations and optimisation and then check to see if they were valid. Another project that we are contemplating is to try and get an "embedded" version of ghc running on our new chips which contain between one and four IBM PowerPC405 processors which are directly connected to the reconfigurable fabric of our chips via IBM's CoreConnect bus. It would be wonderful to have Haskell programs running directly on our chips! Alternatively we may have to come up with

specific requirements for an "embedded lazy functional language" and see to what extent this can be a slight variant of Haskell. For more information about Xilinx's latest chips see `http://www.xilinx.com/virtex2pro/`.
We are now designing Lava circuits with bus interfaces which poses new challenges for us as well as new research opportunities. We plan to design special combinators to facilitate the connection of regular circuits to buses by trying to abstract away as much detail as possible about the bus-interface logic. Bus-interface logic is also very tricky to get right and we plan to try and use external tools to make sure that we comply with the bus interface standard (e.g. make sure that two circuits do not simultaneously try to write to the bus).
There will be a public release of Lava available for download from Xilinx. This will be a ghc binary and will include support for using Lava in the ghc interpreter ghci as well as full binaries for all the Lava libraries.

Further reading:

For more information see `http://www.xilinx.com/labs/lava/` or contact <Satnam.Singh@xilinx.com>

6.2.2 Galois Connections, Inc.

Report by: *Andy Gill*
Galois Connections is a contract engineering and product development software house that uses Haskell as the language of choice. We are a Haskell success story - a profitable startup that is employing around a dozen Haskell engineers.

Further reading:

For more details, goto `http://www.galois.com`, or contact <andy@galois.com>.

6.3 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!

6.3.1 Functional Programming at Yale

Report by: *Henrik Nilsson*
The functional programming group at Yale is using Haskell and general functional language principals to design 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. The languages we have developed are usually based on Functional Reactive Programming (FRP). Particular examples are Frob (Functional Robotics) and FVision (Functional Vision). 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.
FRP is a work in progress: there are many decision points in the FRP design space and we view FRP as a family of languages rather than a specific one. We have recently changed our perspective a bit, emphasizing the notion of signal functions while signals are no longer first class entities. This has a number of notational as well as operational advantages. Moreover, It has enabled us to recast the central ideas from FRP in the setting of John Hughes's arrows framework (see section 3.6.1). The result is AFRP, Arrowized FRP, which is where we currently focus our implementation efforts as far as Haskell-based FRP-implementations are concerned.
Although FRP has traditionally been implemented in Haskell, we have also been looking at direct compilation of FRP programs. We are particularly interested in compilation for resource-limited systems such as embedded controllers.
We have not yet formally released a version of FRP or our FRP-based languages such as Frob or FVision. However, source snapshots for some of our systems are available for downloading ("as is"), and anyone interested in our other systems are encouraged to get in touch with us.
At present, the members of our group are Paul Hudak, John Peterson, Henrik Nilsson, Walid Taha, Antony Courtney, Zhanyong Wan, and Liwen Huang.

Further reading:

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

6.3.2 Functional Programming Research Group at Kingston Business School (Kingston University)

Report by: *Chris Reade*
Application Area: Internet applications
Members:
(Kingston) Chris Reade, Dan Russell, Phil Molyneux, Barry Avery, David Martland
(British Airways) Dominic Steinitz
Contact: Dan Russell <D.Russell@kingston.ac.uk>
This is a relatively new community which has been developing internet applications using advanced language features (functional, typed and higher order). Part of our motivation is to investigate advantages of a functional approach to such application areas, but also to identify areas for further language and library development.
We have built an LDAP client with a web user interface entirely in Haskell (reported at the 3rd Scottish Functional Programming Workshop in August 2001). This has been further developed to include asynchronous processes (using Concurrent Haskell) and a review of robustness issues. We are also extending this work to include SNMP.

Libraries for the LDAP, ASN.1 and BER will be made available as open source very soon.

Further reading:

FP Group: `http://www.kingston.ac.uk/~bs_s075/Research/fpres.html`
Chris Reade: `http://www.kingston.ac.uk/~bs_s075/`

6.3.3 Functional Programming at UKC

Report by: *Claus Reinke*

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

Keith Hanna is continuing development of Vital, a visual interactive implementation of (a subset of) Haskell intended for general use in scientific/financial applications. In Vital, the visual representation of Haskell datatypes is determined by a user-defined stylesheet and the structure/content of data structures can be edited by mouse-based copy-and-paste gestures. An overview of Vital, a web-based simulation and a prototype are available; a new release is planned for later this year.

Axel Simon has just released a Haskell binding for Gtk2. The Gtk toolkit in the most recent version 2.0 features a new List and Edit widget, Unicode and (in the next minor release) Win32 support. Rewritten from scratch, gtk2hs makes all these features available while providing automatic memory management and simpler type and signal handling.

Claus Reinke (yours truly) is still working on the combination of functional programming and virtual worlds (3d animated graphics). A draft paper and presentation (animated 3d, of course!-) from last year's IFL are available, describing a VRML-based version of FunWorlds. But while the initial results still look promising, the frustrating shortcomings of VRML have so far kept me from a proper release. Currently, I'm trying to rebuild on top of HOpenGL instead, for greater flexibility – watch this space, as they say!-)

Chris Ryder is continuing his work on software metrics. The Medina library now contains functionality to write simple metrics and display them in a variety of visualisation styles, mostly using web browsers as the output medium. Chris is currently working on integrating Medina with CVS repositories to enable temporal operations (such as looking at how a metric value changes over time). This is working towards some validation of metrics by looking at the correlation between various metric values for various programs and the change history of those programs.

As reported last time, **Simon Thompson** and **Claus Reinke** have been investigating the potential for *refactoring functional programs*. The project has been granted funding and it now seems that we will have somone in place on the project researcher position early this summer, so that the work can start for real then. Refactoring means changing the structure of existing programs without changing their functionality, and has become popular in the object-oriented and extreme programming communities as a means to achieve continuous evolution of program designs. We want to explore the wealth of functional program transformation research to bring refactoring to Haskell programmers.

Further reading:

FP group:
`http://www.cs.ukc.ac.uk/research/groups/tcs/fp/`
Vital:
`http://www.cs.ukc.ac.uk/people/staff/fkh/Vital/`
Gtk2HS: `http://gtk2hs.sourceforge.net/`
FunWorlds:
`http://www.cs.ukc.ac.uk/people/staff/cr3/FunWorlds/`
Haskell metrics:
`http://www.cs.ukc.ac.uk/people/rpg/cr24/medina/`
Refactoring Functional Programs:
`http://www.cs.ukc.ac.uk/people/staff/sjt/Refactor/`

6.4 Individual Haskellers

The call for contributions for this section asked: "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?).

Amanda Clare <ajc99@aber.ac.uk> writes: I'm currently using Haskell as a tool to write *a data mining program which finds frequent patterns or associations in relational bioinformatics data.* I also use it for *general data processing/ collecting stats/ etc.* I'm doing a PhD in computational biology at Aberystwyth, UK.

The data is structured (relational) - things like "gene X is similar (50%) to gene Y which has a large molecular weight and an alpha helix of length 9 at position 14". Then there are hierarchies wherever you look in biology, for species taxonomy, gene functional class, etc. Even my frequent patterns have common subsections that nicely fit a tree structure. So I don't think it's an unusual task for Haskell, I wouldn't like to handle all these data structures in a C-like language. Also running speed is not really such an issue for me here. If this program runs in 3 weeks rather than 3 days, well, that's okay since the data collection took months (not that I think there would be this much discrepancy anyway). I need the results

but my time spent in programming is limited too. In this university if we take longer than 4 years for a PhD we fail automatically.
Along the way I've had to do lots of "resource usage tuning". I ended up writing down things I should remember to do next time to combat unwanted laziness, in `http://users.aber.ac.uk/ajc99/stricthaskell.html` . It's about the tools and techniques that were useful to me. Tuning is difficult, especially if you're the only person using Haskell in your institution. The mailing lists and profilers, particularly nhc98's excellent profiles, are especially useful! What's missing? I think a Strict Haskell is missing.

Tom Pledger <Tom.Pledger@peace.com> points out that some isolated Haskell users are poorly placed for informal collaboration:
Last year, some of my Haskell-related tinkerings turned from a hobby into an official 1-person research project. My employer regards some key features of the project as trade secrets and potentially patentable. I refrain from asking related questions on the Haskell mailing lists, because it would be

- deceitful, like soliciting answers to homework assignments but in a commercial setting, and
- harmful to my employer's chances of getting a patent, because I'd be provoking people to think up related prior art and release it into the public domain.

Wolfgang Jeltsch (`http://www.wolfgang.jeltsch.net/`) is a student in Computer Science at the Brandenburg University of Technology at Cottbus, Germany. In his free time, he is working on a project called the *Haskell Web Publisher*. The HWP shall be *a software package allowing website implementations in Haskell*. By using Haskell's type system appropriately, several validity and consistency constraints shall be forced by the Haskell compiler. In connection with the HWP, Wolfgang is developing Seaweed, a small library containing utilities concerning XML, the internet and other areas. He has also the vision of adapting the ideas behind HWP for typesetting. This could lead to some kind of TeX replacement where documents are written in Haskell. Wolfgang is looking for people who are interested in helping him realizing HWP and Seaweed. The projects are hosted on SourceForge.net and a paper describing the basic ideas is available via his home page.

Ketil Z. Malde <ketil@ii.uib.no> writes: I'm using Haskell to implement *a bioinformatics algorithm, more specifically sequence (EST) clustering*. As usual, using better algorithms means that I can beat hand-crafted C. I hope :-)
Sorry, there's nothing on my home page yet, but hopefully I'll be able to finish this some time before Summer.

Johannes Waldmann <joe@isun.informatik.uni-leipzig.de> reports on two projects in the context of his university teaching (not only Haskell). Source code for both is available via anon. CVS:

1. *autotool (automatic homework assessment)*

   The system autotool is used at Leipzig University to grade students' homework for courses in theoretical computer science, in particular for exercises on automata, grammars, and recursive functions. The system is implemented with Haskell, and accessed via an email interface.

   The novel aspect of autotool is that students send their solutions as Haskell code, which then gets dynamically loaded into a grading program. That way, the system makes use of advantages of embedded (over interpreted) domain specific languages. In particular, it uses Haskell syntax and typing.

   Strict typing is not only a pedagogical aim. The implied distinction between values and IO actions is crucial for the safe execution of the simulator.

   Project home page (in German): `http://theopc.informatik.uni-leipzig.de/~autotool/`

   Project description (in English): `http://www.informatik.uni-leipzig.de/~joe/pub/draft/hw01.ps`

2. *Modules for Boardgames*

   At the Institute for Informatik, University of Leipzig, we regularily run student programming contests for board games. (In 2001: Connections, in 2002: Philosopher's Phutball.)

   We use a generic game server that is written in Haskell. It uses GHC's Socket and Concurrent library.

   Also, there is a generic Haskell program for Alpha/Beta game tree search. This serves as a default opponent for the students' programs.

   It is straightforward to extend the system for new games. All you need to do is implement

   - a referee (that knows the rules) on the server side,
   - a move generator and evaluation function on the client side,
   - a Java applet (if you want to play directly)

   Contest home page (in German): `http://theopc.informatik.uni-leipzig.de/~joe/phutball/`

Kevin Backhouse (`http://web.comlab.ox.ac.uk/oucl/work/kevin.backhouse/`) is in the final stages of his D.Phil. research on "*Abstract Interpretation of Domain-Specific Embedded Languages*". He sends the following abstract:
Embedded Languages are a promising new approach to the design of Domain Specific Languages (DSLs). Rather than designing new languages from scratch, they are created by writing subroutine libraries in a functional language. The most popular language for this purpose is Haskell, because its higher order functions, polymorphism and laziness make it possible to create languages that are concise and declarative without the overhead of having to write a new compiler or interpreter. Unfortunately, this approach to creating DSLs can make it harder to implement domain specific

error-checking. For example, parsers can be written very easily using combinator libraries such as Swierstra's (`http://www.cs.uu.nl/groups/ST/Software/UU_Parsing/`) or Leijen's (`http://www.cs.uu.nl/~daan/parsec.html`). However, these libraries do not have a mechanism for statically detecting errors such as ambiguity and left-recursion. Such static checking is a distinct advantage of parser generators such as Happy (`http://www.haskell.org/happy/`). Parser combinators are based on LL parsing, so it is a shame that the well-known LL(k) test cannot be applied. This problem is solved in my forthcoming thesis. A general framework (based on abstract interpretation) is also introduced for analysing other embedded languages. See my homepage for more information.
He also writes: "Unfortunately, my thesis is currently still in draft form and it is not on the web yet, but it should be coming soon.".

Mike Thomas <mthomas@gil.com.au> writes: I'm working on small bindings to the MPICH (open source parallel processing by message passing) and Proj (open source map projection) libraries in conjunction with *a Haskell library to read, write, display and process GRASS (an open source geographic information system) mapsets and of doing some geochemical modelling, hopefully with the ability to distribute large map computations with MPICH.* The Windows version of GHC is my compiler of choice.
None of this work is available on the web but more information on GRASS may be found at:
`http://grass.itc.it`
And MPICH:
`http://www-unix.mcs.anl.gov/mpi/mpich/`

6.5 Haskell Spin-Offs

Quite a different kind of Haskell applications are those that take the experience accumulated with Haskell to start something new. While it wouldn't be appropriate to classify the work in such projects as Haskell activities, it is useful to know about such spin-offs. Our first entry in this category, Timber, continues the work originally started on O'Haskell, but deviates further from Haskell itself.

6.5.1 Timber

Report by: *Johan Nordlander*
Timber is a Haskell-related programming language being developed at OHSU (formerly OGI) and Chalmers, as part of the Timber project (`http://www.cse.ogi.edu/PacSoft/projects/Timber/`). Timber aims at being a real-time, reactive, object-oriented, and concurrent programming language that incorporates the full power of pure functional programming via the monadic separation of expressions and commands also utilized by Haskell. Unlike Haskell, however, Timber is based on a strict evaluation semantics, which makes it incorrect to classify Timber simply as a Haskell extension. Still, it shares many of its basic features with Haskell: basic syntax for expressions, bindings and types; a type system based on qualified types (including most Haskell extensions); and pure (although strict) expression evaluation semantics. The main differences to Haskell are:

- A different top-level monad, that supports concurrent reactive objects as the principal structuring concept.
- A built-in notion of time and a time-constrained reaction.
- Named records and subtyping.

Timber also represents the continued development of O'Haskell (`http://www.cs.chalmers.se/~nordland/ohaskell/`), a language in which many of the distinguishing features of Timber where first introduced. Apart from the switch to strict evaluation, the major change in Timber w.r.t. O'Haskell is the introduction of real-time constraints into the programming model. Moreover, unlike O'Haskell, Timber fully integrates its support for subtyping into the qualified types framework.
An interpreter that illustrates some of the core Timber concepts exists in the shape of an extended O'Haskell interpreter (`http://www.cs.chalmers.se/~nordland/ohugs/`, latest CVS version). A compiler for the full Timber language is currently under development, and an initial release is planned for later this year.
Current participants include Andrew Black, Magnus Carlsson, Mark Jones, Dick Kieburtz, Johan Nordlander, and Björn von Sydow.
Contact: Johan Nordlander <nordland@cs.chalmers.se>