

# Haskell Communities and Activities Report

<http://tinyurl.com/haskcar>

Twenty-Fifth Edition — November 2013

Mihai Maruseac, Janis Voigtländer (eds.)

Andreas Abel	Heinrich Apfelmus	Daniel Austin
Emil Axelsson	Doug Beardsley	Jean-Philippe Bernardy
Jeroen Bransen	Joachim Breitner	Erik de Castro Lopo
Roman Chepyaka	Olaf Chitil	Alberto Gómez Corona
Duncan Coutts	Atze Dijkstra	Péter Diviánszky
Dennis Felsing	Julian Fleischer	Michal J. Gajda
Andy Georges	Andrew Gibiansky	Brett G. Giles
Andy Gill	Jurriaan Hage	Greg Hale
Bastiaan Heeren	Sylvain Henry	PÁLI Gábor János
Paul Hudak	Bob Ippolito	Philipp Kant
Robin KAY	Anton Kholomiov	Oleg Kiselyov
Edward Kmett	Eric Kow	Nickolay Kudasov
Ben Lippmeier	Andres Löh	Rita Loogen
Boris Lykah	Ian Lynagh	Christian Maeder
José Pedro Magalhães	Ketil Malde	Mihai Maruseac
Kazutaka Matsuda	Dino Morelli	JP Moresmau
Ben Moseley	Jürgen Nicklisch-Franken	Tom Nielsen
Rishiyur Nikhil	Kiwamu Okabe	Jens Petersen
Simon Peyton Jones	David Sabel	Uwe Schmidt
Martijn Schrage	Tom Schrijvers	Jeremy Shaw
Christian Höner zu Siederdisen	Michael Snoyman	Andrei Soare
Doaitse Swierstra	Wouter Swierstra	Bernhard Urban
Alessio Valentini	Adam Vogt	Daniel Wagner
Kazu Yamamoto	Edward Z. Yang	Brent Yorgey
	Alan Zimmerman	

## Preface

This is the 25th edition of the Haskell Communities and Activities Report. As usual, fresh entries are formatted using a blue background (there are a lot of those by a single person this time around; guess by whom), while updated entries have a header with a blue background. Entries on which no new activity has been reported for a year or longer have been dropped completely. Please do revive such entries next time if you do have news on them.

A few words from Janis: This is the last instance of HCAR in which I have a hand as editor. It's been a few years, and I am happy that Mihai has stepped in to continue the work of putting together editions of the report, and will be joined by Alejandro Serrano Mena next time around. Thanks to everybody for support in the last years. And thanks to Mihai and Alejandro for taking over; now it's your turn to evoke guilty feelings in your fellow Haskellers: <http://ro-che.info/ccc/16.html>.

A call for new HCAR entries and updates to existing ones will be issued on the Haskell mailing lists in April. Now enjoy the current report and see what other Haskellers have been up to lately. Any feedback is very welcome, as always.

Janis Voigtländer, University of Bonn, Germany  
Mihai Maruseac, University of Massachusetts Boston, US  
<[hcar@haskell.org](mailto:hcar@haskell.org)>

# Contents

<b>1</b>	<b>Community</b>	<b>7</b>
1.1	Haskell' — Haskell 2014 . . . . .	7
1.2	Haskellers . . . . .	7
<b>2</b>	<b>Books, Articles, Tutorials</b>	<b>8</b>
2.1	The Monad.Reader . . . . .	8
2.2	Oleg's Mini Tutorials and Assorted Small Projects . . . . .	8
2.3	Agda Tutorial . . . . .	9
2.4	School of Haskell . . . . .	9
<b>3</b>	<b>Implementations</b>	<b>10</b>
3.1	The Glasgow Haskell Compiler . . . . .	10
3.2	Ajhc Haskell Compiler . . . . .	13
3.3	UHC, Utrecht Haskell Compiler . . . . .	14
3.4	Specific Platforms . . . . .	14
3.4.1	Haskell on FreeBSD . . . . .	14
3.4.2	Debian Haskell Group . . . . .	15
3.4.3	Fedora Haskell SIG . . . . .	15
<b>4</b>	<b>Related Languages and Language Design</b>	<b>16</b>
4.1	Agda . . . . .	16
4.2	MiniAgda . . . . .	16
4.3	Disciple . . . . .	16
4.4	Ermine . . . . .	17
<b>5</b>	<b>Haskell and . . .</b>	<b>18</b>
5.1	Haskell and Parallelism . . . . .	18
5.1.1	Eden . . . . .	18
5.1.2	Parallel GHC project . . . . .	19
5.1.3	speculation . . . . .	20
5.2	Haskell and the Web . . . . .	20
5.2.1	WAI . . . . .	20
5.2.2	Warp . . . . .	20
5.2.3	Holumbus Search Engine Framework . . . . .	21
5.2.4	Happstack . . . . .	21
5.2.5	Mighttpd2 — Yet another Web Server . . . . .	22
5.2.6	Yesod . . . . .	22
5.2.7	Snap Framework . . . . .	23
5.2.8	Sunroof . . . . .	24
5.2.9	MFlow . . . . .	24
5.3	Haskell and Compiler Writing . . . . .	25
5.3.1	MateVM . . . . .	25
5.3.2	UUAG . . . . .	25
5.3.3	LQPL — A Quantum Programming Language Compiler and Emulator . . . . .	26
5.3.4	free — Free Monads . . . . .	27
5.3.5	bound — Making De Bruijn Succ Less . . . . .	27
<b>6</b>	<b>Development Tools</b>	<b>28</b>
6.1	Environments . . . . .	28
6.1.1	Haskell IDE From FP Complete . . . . .	28
6.1.2	EclipseFP . . . . .	28
6.1.3	Ariadne . . . . .	29

6.1.4	ghc-mod — Happy Haskell Programming	29
6.1.5	HEAT: The Haskell Educational Advancement Tool	30
6.1.6	HaRe — The Haskell Refactorer	30
6.1.7	IHaskell: Haskell for Interactive Computing	30
<b>6.2</b>	<b>Code Management</b>	<b>31</b>
6.2.1	Darcs	31
6.2.2	DarcsWatch	32
6.2.3	cab — A Maintenance Command of Haskell Cabal Packages	32
<b>6.3</b>	<b>Deployment</b>	<b>32</b>
6.3.1	Cabal and Hackage	32
6.3.2	Stackage: the Library Dependency Solution	33
6.3.3	standalone-haddock	34
<b>6.4</b>	<b>Others</b>	<b>34</b>
6.4.1	lhs2TeX	34
6.4.2	ghc-heap-view	35
6.4.3	ghc-vis	35
6.4.4	Hat — the Haskell Tracer	35
6.4.5	Tasty	36
<b>7</b>	<b>Libraries, Applications, Projects</b>	<b>37</b>
<b>7.1</b>	<b>Language Features</b>	<b>37</b>
7.1.1	Conduit	37
7.1.2	lens	37
7.1.3	folds	38
7.1.4	machines	38
7.1.5	exceptions	38
7.1.6	tables	38
<b>7.2</b>	<b>Education</b>	<b>39</b>
7.2.1	Exercism: crowd-sourced code reviews on daily practice problems	39
7.2.2	Talentbuddy	39
7.2.3	Holmes, Plagiarism Detection for Haskell	39
7.2.4	Interactive Domain Reasoners	39
<b>7.3</b>	<b>Parsing and Transforming</b>	<b>40</b>
7.3.1	FliPpr	40
7.3.2	epub-metadata	40
7.3.3	Utrecht Parser Combinator Library: uu-parsinglib	41
7.3.4	Grammar Products	42
7.3.5	HERMIT	42
7.3.6	haskell-names	43
7.3.7	haskell-packages	43
7.3.8	parsers	43
7.3.9	trifecta	44
<b>7.4</b>	<b>Generic and Type-Level Programming</b>	<b>44</b>
7.4.1	Optimising Generic Functions	44
7.4.2	traverse-with-class	44
7.4.3	constraints	45
<b>7.5</b>	<b>Mathematics</b>	<b>45</b>
7.5.1	Rlang-QQ	45
7.5.2	order-statistics	45
7.5.3	bed-and-breakfast	45
7.5.4	Eliminating Redundancies in Linear Systems	45
7.5.5	linear	46
7.5.6	algebra	46
7.5.7	semigroups and semigroupoids	46
7.5.8	Arithmetics packages (Edward Kmett)	46
7.5.9	ad	47
7.5.10	integration	47
7.5.11	categories	47

7.5.12	contravariant	47
7.5.13	bifunctors	48
7.5.14	profunctors	48
7.5.15	comonad	48
7.5.16	recursion-schemes	48
7.5.17	kan-extensions	49
<b>7.6</b>	<b>Data Types and Data Structures</b>	<b>49</b>
7.6.1	HList — A Library for Typed Heterogeneous Collections	49
7.6.2	Persistent	49
7.6.3	Groundhog	50
7.6.4	reflection	50
7.6.5	tag-bits	51
7.6.6	hyperloglog	51
7.6.7	concurrent-supply	51
7.6.8	hybrid-vectors	51
7.6.9	lca	51
7.6.10	heaps	51
7.6.11	sparse	52
7.6.12	compressed	52
7.6.13	charset	52
7.6.14	Convenience types (Edward Kmett)	52
<b>7.7</b>	<b>User Interfaces</b>	<b>53</b>
7.7.1	HsQML	53
7.7.2	LGtk: Lens-based Gtk API	53
7.7.3	Gtk2Hs	53
7.7.4	Haskell-EFL binding	54
7.7.5	threepenny-gui	54
7.7.6	reactive-banana	55
<b>7.8</b>	<b>Graphics and Audio</b>	<b>55</b>
7.8.1	diagrams	55
7.8.2	csound-expression	56
7.8.3	Chordify	57
7.8.4	Euterpea	57
<b>7.9</b>	<b>Text and Markup Languages</b>	<b>58</b>
7.9.1	Haskell XML Toolbox	58
7.9.2	epub-tools (Command-line epub Utilities)	59
7.9.3	lens-aeson	59
7.9.4	hyphenation	59
<b>7.10</b>	<b>Natural Language Processing</b>	<b>59</b>
7.10.1	NLP	59
7.10.2	GenI	60
<b>7.11</b>	<b>Bioinformatics</b>	<b>61</b>
7.11.1	ADPfusion	61
7.11.2	<i>Ab-initio</i> electronic structure in Haskell	61
7.11.3	Semi-Classical Molecular Dynamics in Haskell	62
7.11.4	Biohaskell	63
7.11.5	arte-ephys: Real-time electrophysiology	64
<b>7.12</b>	<b>Embedding DSLs for Low-Level Processing</b>	<b>64</b>
7.12.1	Feldspar	64
7.12.2	Kansas Lava	64
<b>7.13</b>	<b>Others</b>	<b>65</b>
7.13.1	General framework for multi-agent systems	65
7.13.2	ersatz	65
7.13.3	Clckwrks	65
7.13.4	FNISstash	66
7.13.5	arbtt	66
7.13.6	java-bridge	66

<b>8</b>	<b>Commercial Users</b>	<b>67</b>
<b>8.1</b>	Well-Typed LLP . . . . .	<b>67</b>
<b>8.2</b>	Bluespec Tools for Design of Complex Chips and Hardware Accelerators . . . . .	<b>67</b>
<b>8.3</b>	Industrial Haskell Group . . . . .	<b>68</b>
<b>8.4</b>	Barclays Capital . . . . .	<b>68</b>
<b>8.5</b>	Oblomov Systems . . . . .	<b>69</b>
<b>8.6</b>	OpenBrain Ltd. . . . .	<b>69</b>
<b>9</b>	<b>Research and User Groups</b>	<b>70</b>
<b>9.1</b>	Haskell at Eötvös Loránd University (ELTE), Budapest . . . . .	<b>70</b>
<b>9.2</b>	Artificial Intelligence and Software Technology at Goethe-University Frankfurt . . . . .	<b>70</b>
<b>9.3</b>	Functional Programming at the University of Kent . . . . .	<b>71</b>
<b>9.4</b>	Formal Methods at DFKI and University Bremen and University Magdeburg . . . . .	<b>71</b>
<b>9.5</b>	Haskell at Universiteit Gent, Belgium . . . . .	<b>72</b>
<b>9.6</b>	Haskell in Romania . . . . .	<b>73</b>
<b>9.7</b>	fp-syd: Functional Programming in Sydney, Australia . . . . .	<b>74</b>
<b>9.8</b>	Functional Programming at Chalmers . . . . .	<b>74</b>
<b>9.9</b>	Functional Programming at KU . . . . .	<b>76</b>
<b>9.10</b>	Ghent Functional Programming Group . . . . .	<b>76</b>
<b>9.11</b>	Odessa Haskell User Group . . . . .	<b>77</b>
<b>9.12</b>	Functional Programming at Utrecht University . . . . .	<b>77</b>

# 1 Community

## 1.1 Haskell' — Haskell 2014

---

Report by:	Ian Lynagh
Participants:	Carlos Camarão, Iavor Diatchki, Bas van Dijk, Ian Lynagh, John Meacham, Neil Mitchell, Ganesh Sittampalam, David Terei, Henk-Jan van Tuyl

---

Haskell' is an ongoing process to produce revisions to the Haskell standard, incorporating mature language extensions and well-understood modifications to the language. New revisions of the language are expected once per year.

The Haskell 2014 committee has now formed, and we would be delighted to receive your proposals for changes to the language. Please see <http://hackage.haskell.org/trac/haskell-prime/wiki/Process> for details on the proposal process.

The committee will meet 4 times a year, to consider proposals completed before:

- o 1st August
- o 1st November
- o 1st February
- o 1st May

So if you have been meaning to put the finishing touches to a proposal, then we would encourage you to do so by the end of July!

The source for the Haskell report will be updated as proposals are accepted, but new versions of the standard will only be released once a year, during January.

## 1.2 Haskellers

---

Report by:	Michael Snoyman
Status:	experimental

---

Haskellers is a site designed to promote Haskell as a language for use in the real world by being a central meeting place for the myriad talented Haskell developers out there. It allows users to create profiles complete with skill sets and packages authored and gives employers a central place to find Haskell professionals.

Since the May 2011 HCAR, Haskellers has added polls, which provides a convenient means of surveying a large cross-section of the active Haskell community. There are now over 1300 active accounts, versus 800 one year ago.

Haskellers remains a site intended for all members of the Haskell community, from professionals with 15 years experience to people just getting into the language.

### Further reading

<http://www.haskellers.com/>

## 2 Books, Articles, Tutorials

### 2.1 The Monad.Reader

Report by: Edward Z. Yang

There are many academic papers about Haskell and many informative pages on the HaskellWiki. Unfortunately, there is not much between the two extremes. That is where The Monad.Reader tries to fit in: more formal than a wiki page, but more casual than a journal article.

There are plenty of interesting ideas that might not warrant an academic publication—but that does not mean these ideas are not worth writing about! Communicating ideas to a wide audience is much more important than concealing them in some esoteric journal. Even if it has all been done before in the Journal of Impossibly Complicated Theoretical Stuff, explaining a neat idea about “warm fuzzy things” to the rest of us can still be plain fun.

The Monad.Reader is also a great place to write about a tool or application that deserves more attention. Most programmers do not enjoy writing manuals; writing a tutorial for The Monad.Reader, however, is an excellent way to put your code in the limelight and reach hundreds of potential users.

Since the last HCAR there has been one new issue, featuring tutorials on generalized algebraic data types, monad transformers, and approximating NP-complete problems with monoids.

#### Further reading

<http://themonadreader.wordpress.com/>

### 2.2 Oleg’s Mini Tutorials and Assorted Small Projects

Report by: Oleg Kiselyov

The collection of various Haskell mini tutorials and assorted small projects (<http://okmij.org/ftp/Haskell/>) has received three additions:

#### Many ways to the fix-point combinator beside the value recursion

Value-level recursion in Haskell is built-in since definitions are implicitly recursive. Had the value-level recursion been unavailable, we could still safely define the polymorphic fix-point combinator in Haskell – in

*five* different ways. Iso-recursive data types are a well-known way to *fix*. Less known is using type classes or families. The lazy ST approach is most puzzling: the reading of a reference cell appears to occur in pure code.

Uncharitably speaking, Haskell, taken as a logic, is inconsistent in more than two ways.

<http://okmij.org/ftp/Haskell/types.html#fix>

#### In defense of UndecidableInstances

Class instance and type family instance declarations are subject to a set of strict conditions that ensure the termination of resolving class constraints or normalizing type function applications. The conditions are quite constrictive, and GHC offers a way to lift them, with the LANGUAGE pragma `UndecidableInstances`. The pragma is shunned however. There are good reasons for that attitude: the pragma lets us write instances that send the type checker into infinite loop (albeit only potentially: the termination is insured by the recursion depth limit).

However, the decidability conditions are sufficient but by no means necessary to ensure the decidability of type normalization. There are patently decidable type programs that require `UndecidableInstances`. This article shows a simple example, and argues for more acceptance towards `UndecidableInstances`. The extension never causes the type-checker to accept a program that ‘goes wrong.’ It should not be categorically stigmatized.

`DecidableInstances` are quite like the primitive recursion criterion: all primitive recursive functions surely terminate; non-primitive recursive functions generally don’t. Still there are many classes of non-primitive recursive functions that are total.

<http://okmij.org/ftp/Haskell/TypeClass.html#undecidable-inst-defense>

#### Dynamic epistemic logic puzzles

The article presents a simplistic model-theoretic framework to solve the puzzles like the following:

Anne, Bill and Cath each have a positive natural number written on their foreheads. They can only see the foreheads of others. One of the numbers is the sum of the other two. All the previous is common knowledge. The following truthful conversation takes place:

**Anne** I don’t know my number.

**Bill** I don’t know my number.

**Cath** I don’t know my number.



**Anne** I now know my number, and it is 50.

What are the numbers of Bill and Cath?

We encode the statement of the problem as a filter on possible worlds. The possible worlds consistent with the statement of the problem are the solutions. “Agent A does not know proposition  $\phi$ ” is interpreted as the statement that for all worlds consistent with the propositions that A currently knows,  $\phi$  is true in some worlds but false in the others.

<http://okmij.org/ftp/Algorithms.html#dyn-epistemology>

## 2.3 Agda Tutorial

Report by:	Péter Diviánszky
Participants:	Ambrus Kaposi, students at ELTE IK
Status:	experimental

Agda may be the next programming language to learn after Haskell. Learning Agda gives more insight into the various type system extensions of Haskell, for example.

The main goal of the tutorial is to let people explore programming in Agda without learning theoretical background in advance. Only secondary school mathematics is required for the tutorial.

I currently work on the correction and completion of the existing material.

### Further reading

<http://people.inf.elte.hu/divip/AgdaTutorial/Index.html>

## 2.4 School of Haskell

Report by:	Michael Snoyman
Participants:	Edward Kmett, Simon Peyton Jones and others
Status:	active

Strides are being made to drive greater Haskell adoption. One resource helping with this push is the School of Haskell at FP Complete. The School of Haskell contains tutorials, courses, and articles created by both the Haskell community and the developers at FP Complete. Courses for all levels of developers are available and since going live in early 2013, it has benefited immensely from a lot of excellent content provided by the Haskell community.

The School of Haskell is an excellent resource for Haskell developers looking to share their projects and to become more proficient with Haskell. So far 1400 tutorials have been created and 288 have been officially published. Some very notable authors including, Edward Kmett, Michael Snoyman, and Simon Peyton Jones have contributed tutorials. To date, the school of Haskell has had nearly 25k unique visitors.

All Haskell programmers are encouraged to visit the School of Haskell and to contribute their ideas and projects. This is another opportunity to showcase the virtues of Haskell and the sophistication and high level thinking of the Haskell community.

### Further reading

<https://www.fpcomplete.com/school>

## 3 Implementations

### 3.1 The Glasgow Haskell Compiler

Report by:	Simon Peyton Jones
Participants:	many others

A big event occurred earlier this year in July: the news that Ian Lynagh would be leaving Well-Typed — and consequently, GHC HQ — to move onto new challenges. Ian worked on GHC for 7 years full-time, helped write our new build system, redesigned large parts of the compiler, and fixed innumerable other issues over the course of his work with us. According to some statistics for the past 7 years or so [Contr], Ian quite clearly is one of the biggest contributors we’ve ever had. And we should take the time to say what is rightfully deserved: **thank you for everything you’ve done for us, Ian!**

While Ian has not completely disappeared, GHC and Haskell are not his day-job anymore. As a result, this means everyone — including you, our friend and reader — have a huge opportunity to help continue making GHC even better. Luckily, the past few months of activity points towards a tremendous surge in community involvement. GHC is a community project, and the community is what keeps us afloat. **A great thanks goes to all those who have helped us come so far!**

In light of this, Well-Typed has added two people to GHC HQ to keep things moving, and help the development process going forward: **Edsko de Vries** and **Austin Seipp**.

The GHC 7.8 release is in its final stages, and will be released in late November according to our plans. There’s a tremendous amount of exciting changes coming very soon, described below.

#### Source language and type system:

**Type natural solver.** Iavor Diatchki implemented a basic constraint solver for the type naturals extension, meaning that GHC can now infer and understand basic identities such as  $(x+2) \sim 5$ , which implies  $x = 3$ . The constraint solver does only basic things for now; there is more to come.

**Closed type families.** Richard Eisenberg and co. have implemented support for closed type families in GHC, allowing you to write a type family where no instances can be made beyond the ones in the definition (wiki page [CTF-wiki] and paper [CTF-paper]). This allows a host of new programs to be expressed, as we now know certain

invariants can hold. For example, we may now write:

```
type family Flip p :: *
type family Flip a where
  Flip Even = Odd
  Flip Odd = Even
```

to express that `Flip` may have no more instances written, meaning nefarious users can no longer write silly instances like `Flip Bool = Int`.

**Role support.** Richard Eisenberg implemented support for role checking in GHC, fixing a long standing bug where `GeneralizedNewtypeDeriving` could be used to derive unsafe instances for a `newtype` (wiki [Roles-wiki] and paper [Roles-paper]).

**New and improved I/O manager.** Earlier this year, Andreas Voellmy and Kazu Yamamoto worked on a host of improvements to our I/O manager, making it scale significantly better on multicore machines. Since then, it’s seen some other performance tweaks, and many bugfixes. As a result, the new I/O manager should scale linearly up to about 40 cores. Andreas reports their McNettle Software-defined-network (SDN) implementation can now achieve over twenty million connections per second, making it the fastest SDN implementation around — an incredible feat!

**Type Holes.** Thijs Alkemade and Simon PJ got an implementation of `TypeHoles` in GHC, meaning it’s possible to tell GHC there is a ‘hole’ in a program, and have the compiler spit out an error stating what types are in scope. As a trivial example

```
Prelude> :set -XTypeHoles
Prelude> let f :: a -> a; f x = _

<interactive>:6:24:
  Found hole '_' with type: a
  Where: 'a' is a rigid type variable
         bound by the type signature for
         f :: a -> a at <interactive>:6:10
  Relevant bindings include
    x :: a (bound at <interactive>:6:20)
    f :: a -> a (bound at
      <interactive>:6:18)
  In the expression: _
  In an equation for 'f': f x = _
```

GHC now tells us that the term `f` has a hole of type `a`, and there is a term `x :: a` in scope. So the definition

is clear:  $f\ x = x$ . Holes are originally a concept borrowed from Agda, and we hope they will be useful to Haskell programmers too!

**New Template Haskell.** Geoffrey Mainland implemented support for New Template Haskell, fixing a lot of long-standing bugs with the TH implementation, while making it significantly more expressive, including support for typed quotes, pattern splices and more. This allows us to write for example a typed, staged power function. For details and examples, you can see Geoff’s blog [NewTH1] and the GHC wiki pages describing the design [NewTH2].

**Newtype coercions.** Joachim Breitner spent time earlier this year implementing a new, fancy **Coercible** design in GHC, which helps eliminate the cases in which newtypes are ‘not free’. This occurs for example, when we say something like `newtype Age = MkAge Int` and then `map MkAge [0..10]` — while **newtype** is supposed to be free, this particular example has a runtime cost! While **Coercible** is still in flux [NTC], we hope it will be useful as a ‘preview technology’ in the 7.8 release.

A small menagerie of various other language improvements and extensions, including:

- `-XOverloadedLists` which allows overloading list literals (Achim Krause, George Giorgidze, Weijers Jeroen)
- `-XNumDecimals` allowing a compact floating-point syntax for integrals, e.g. `1.2e6 :: Integer` (Shachaf Ben-Kiki)
- Support for empty case statements (via `-XEmptyCase`, by Richard Eisenberg)
- `-XIncoherentInstances` became even more liberal (Joachim Breitner)
- A new pragma for specifying the minimal complete definition of a typeclass, via `{-# MINIMAL #-}` (Twan van Laarhoven)

### Back end and runtime system:

**New code generator.** As previously reported, the New Code Generator is live and switched on by default. There have been a host of bugfixes and stability improvements, meaning it should be solid for the 7.8 release.

**SSE/AVX support.** Geoffrey Mainland implemented support for SSE/AVX intrinsics in the compiler backend, making it possible to exploit hardware-accelerated SIMD operations in your code on Intel/AMD machines. It’s currently only enabled for the LLVM backend, however.

**LLVM back end.** Peter Wortmann spent time earlier this year doing a significant refactoring of the LLVM

backend, which means it should be easier in the future to extend the backend and perform long-term maintenance.

**Code generation.** Jan Stolarek had an internship at Microsoft Research during the summer, and as part of this he implemented an array of improvements to the code generator and backend, including a new loopification pass to turn tail-recursive calls into loops, and a refactoring of our **Bool** based primops to return unboxed `Int#` values (making them much faster, with a sizeable performance improvement in some cases [PrimBool].)

**Unloading object code.** Simon Marlow implemented support for unloading object code at runtime in the GHC linker. Previously, while GHC’s linker could load object code dynamically, there was no facility to unload it — meaning long running applications would continuously suffer a memory leak as they reloaded more code.

**Library constructors.** Edward Yang implemented support for running library constructors in GHCi, making it possible to use foreign libraries which depend on constructors being run at load time.

**Atomic and prefetch primops.** There are new primops for all backends, including new atomic memory operations (by Ryan Newton) and support for low-level prefetch instructions in the processor, allowing you to guide cache decisions (by Carter Schonwald.)

### Frontend, build-system, and miscellaneous changes:

**Dynamic-by-default.** In 7.8, we’re hoping to make GHCi use the system linker by default on supported platforms, eliminating a host of bugs in the current home-grown linker. Eventually we hope to remove the old linker completely. Until then, GHC now supports compiling files statically and dynamically at the same time (with the `-dynamic-too` flag,) meaning you can switch between static/dynamic builds much more easily.

**Compiler hooks.** Luite Stegeman and Edsko de Vries did a significant amount of work to improve hooking support in the GHC API. This new API makes it possible for users to plug in their own pipeline machinery to the compiler, suitable for implementing new frontend features (like QuasiQuoting hooks) or new backends (like a JavaScript target, as part of the new GHCJS.)

**Parallel `--make`.** As part of the haskell.org 2013 GSoC, Patrick Palka implemented a new parallel compilation driver, a long-requested feature. This allows GHC to build multiple modules in parallel when using `--make` by adding a `-j` flag, while having almost no overhead in the single-threaded case.

**Clang support.** Austin Seipp added support for GHC to use Clang as the C compiler, instead of GCC. While the work is still rough, it should be a boon to modern OS X users, who no longer have GCC available by default.

**iOS support and cross compilation.** After many years of work by Ian, Stephen Blackheath, Gabor Greif and friends Luke Iannini and Maxwell Swadling, GHC now has full support for iOS cross-compilation. As of GHC 7.8, you'll really be able to write iOS apps in your favorite programming language!

**Better ARM support.** Ben Gamari and Austin Seipp have some final changes to make for the ARM story, which should make it significantly more stable and usable, including GHCi support. We hope to have ARMv7 binary releases for GHC 7.8 available for download.

#### Future plans:

After the 7.8 release, there are some improvements scheduled we plan on integrating:

**Applicative as a superclass of Monad.** A long-standing proposal, GHC 7.10 will finally make `Applicative` a superclass of `Monad`. GHC 7.8 features warnings to ensure users know where their code will break as a result of this API change.

**Kinds without data.** Trevor Elliott, Eric Mertens, and Iavor Diatchki have begun implementing support for 'data kind' declarations, described in more detail on the GHC wiki [KD]. The idea is to allow a new form of declaration that introduces a new kind, whose members are described by the (type) constructors in the declaration. This is similar to promoting data declarations, except that no new value-level-constructors are declared, and it also allows the constructors to mention other kinds that do not have corresponding type-level representation (e.g., `*`).

**Overloaded record fields.** Adam Gundry implemented the overloaded records field proposal as part of the Haskell.org 2013 GSoC. This work will make it possible to not only have overloaded record field names, but also enable a wealth of other nice features, like polymorphic update/lookup, and automatically turning record fields into lens ( $\rightarrow$  7.1.2). More detail can be found on the GHC wiki [ORF].

**Pattern synonyms.** Gergő Érdi worked on an implementation of pattern synonyms for GHC, which will finally give us the power to abstract over patterns and give them names. While the design is not final (see the wiki for details[PS]), the results look promising, and will hopefully fix a long-standing 'abstraction hole' in the term language for Haskell.

**Explicit Type Application.** Stephanie Weirich, Richard Eisenberg and Hamidhasan Ahmed have been working on adding explicit type applications to GHC. This allows the programmer to specify the types that should be instantiated for arguments to a function application, where normally they would be inferred. While this capability already exists in GHC's internal language, System FC — indeed, every FC-pro program has function application with explicitly applied types — it has not been available in Haskell itself. While a lot of the syntax and design is not quite final, there are some details about the design available on the wiki [TA].

**Git management changes.** For a long time, GHC has used a loosely coupled set of repositories during development. However, as we've added more contributors, this practice has become increasingly problematic, preventing us from using useful tools like `git bisect` to track down bugs. Our plans after the 7.8 release are to sort this out, and hopefully have stable, reproducible GHC builds for all.

**Continuous Integration overhaul.** GHC has historically been very spotty about continuous integration. Part of the problem is our git management, mentioned above. In the 7.10 timeframe, we hope to spend a significant amount of time overhauling our buildbots and machines. To this end, Rackspace has graciously donated services as part of its Open Source support initiatives.

But we're not sure what else might happen. It's a great time to step up to the plate and do something fun!

#### Development updates, joining in and a big Thank You!

In the past several months, GHC has seen a surge of community involvement, and a great deal of new contributors. We now have 37 committers, with **14 added in 2013 alone** — it's an exciting time to help out!

Amongst those who have rolled up their sleeves and helped us recently:

- **Herbert Valerio Riedel** has stepped up to the plate to help maintain <http://ghc.haskell.org>, all of our git repositories, and has become increasingly involved in library maintenance and development.
- **Takano Akio** has submitted many great bug reports and patches recently, and has a keen eye for reproducing challenging or tricky bugs.
- **Reid Barton** has submitted some excellent patches to fix some awkward bugs, and helps us as we move towards Android cross compiler support.
- **Kazu Yamamoto** helped Andreas Voellmy in the improvements to the I/O manager, and has been extremely helpful in pointing out and reproducing bugs, and helping us find solutions.

- **Joachim Breitner** implemented Coercible, but has also stepped up to merge patches, fix more bugs, and help set up some Continuous-Integration for GHC.
- **Carter Schonwald** has been involved in talking with many who wish to get involved and inspiring them to get their hands dirty!
- **Krzysztof Gogolewski** has fixed many bugs, and like Takano, has helped reproduce and track down issues with fantastic test cases.
- **Jan Stolarek** has helped improve the documentation of the low-level parts of the compiler, as well as implementing optimizations for us too.
- **Muhaimin Ahsan** has been helping with infrastructure, writing patches, and generally offering help to us who are already busy.

As ever, there is a ton of stuff in the future for us to do. If you want something done — don't wait, it might take a while. You should join us instead!

#### Links:

- [Contr], <https://github.com/ghc/ghc/graphs/contributors>
- [CTF-wiki], Closed type families with overlapping equations — <http://ghc.haskell.org/trac/ghc/wiki/NewAxioms>
- [CTF-paper], Closed type families with overlapping equations, Richard Eisenberg, Dimitrios Vytiniotis, Simon Peyton Jones, Stephanie Weirich, POPL 2014 — <http://research.microsoft.com/en-us/um/people/simonpj/papers/ext-f/>
- [KD], Kinds without Data — <http://ghc.haskell.org/trac/ghc/wiki/GhcKinds/KindsWithoutData>
- [NewTH1], Runtime codegen with typed Template Haskell — <http://gmainland.blogspot.com/2013/05/type-safe-runtime-code-generation-with.html>
- [NewTH2], Major proposed Template Haskell revision — <http://ghc.haskell.org/trac/ghc/wiki/TemplateHaskell/BlogPostChanges>
- [NTC], Newtype wrappers — <http://ghc.haskell.org/trac/ghc/wiki/NewtypeWrappers>
- [PrimBool], New comparison primitives — <http://ghc.haskell.org/trac/ghc/wiki/PrimBool>
- [ORF], Overloaded record fields — <http://ghc.haskell.org/trac/ghc/wiki/Records/OverloadedRecordFields/Plan>
- [PS], Pattern synonyms — <http://ghc.haskell.org/trac/ghc/wiki/PatternSynonyms>
- [Roles-wiki], Roles — <http://ghc.haskell.org/trac/ghc/wiki/Roles>
- [Roles-paper], Generative Type Abstraction and Type-level Computation, Stephanie Wirich, Dimitrios Vytiniotis, Simon Peyton Jones, and Steve Zdancewic, POPL 2011 — <http://research.microsoft.com/en-us/um/people/simonpj/papers/ext-f/>
- [TA], Explicit type application — <http://ghc.haskell.org/trac/ghc/wiki/ExplicitTypeApplication>

## 3.2 Ajhc Haskell Compiler

Report by:	Kiwamu Okabe
Participants:	John Meacham, Hiroki MIZUNO, Hidekazu SEGAWA
Status:	experimental

### What is it?

Ajhc is a Haskell compiler, and acronym for “A fork of jhc”.

Jhc (<http://repetae.net/computer/jhc/>) converts Haskell code into pure C language code running with jhc's runtime. And the runtime is written with 3000 lines (include comments) pure C code. It's a magic!

Ajhc's mission is to keep contribution to jhc in the repository. Because the upstream author of jhc, John Meacham, can't pull the contribution speedily. (I think he is too busy to do it.) We should feedback jhc any changes. Also Ajhc aims to provide the Metasepi project with a method to rewrite NetBSD kernel using Haskell. The method is called Snatch-driven development.

Ajhc is, so to speak, an accelerator to develop jhc.

### What is new?

Runtime:

Ajhc now has thread-safe and reentrant runtime. The runtime supports pthread and custom thread that can shape forkOS API on tiny CPU with ChibiOS/RT. (<https://github.com/metasepi/chibios-arafura>)

GC:

New Erlang style GC. It means Ajhc's Haskell context has own GC heap. Also new GC can run on tiny CPU such as Cortex-M3 with 64kB RAM.

Document:

We have translated Jhc User's Manual into Japanese. The translating into the other language will be also easy, because the translators are using gettext. (<https://github.com/ajhc/ajhc/blob/arafura/po/ja.po>)

### Demonstrations

<http://www.youtube.com/watch?v=n6cepTfnFoo>

The touchable cube application is written with Haskell and compiled by Ajhc. In the demo, the application is broken by ndk-gdb debugger when running GC. You could watch the demo source code at <https://github.com/ajhc/demo-android-ndk>.

<http://www.youtube.com/watch?v=C9JsJXWyajQ>

The demo is running code that compiled with Ajhc on Cortex-M3 board, mbed. It's a simple RSS reader for reddit.com, showing the RSS titles on Text LCD panel. You could watch the demo detail and source code at <https://github.com/ajhc/demo-cortex-m3>.

<http://www.youtube.com/watch?v=zkSy0ZroRIs>

The demo is running Haskell code without any OS. Also the clock exception handler is written with Haskell.

## Usage

You can install Ajhc from Hackage.

```
$ cabal install ajhc
$ ajhc --version
ajhc 0.8.0.9 (9c264872105597700e2ba403851cf3b236cb1646)
compiled by ghc-7.6 on a x86_64 running linux
$ echo 'main = print "hoge"' > Hoge.hs
$ ajhc Hoge.hs
$ ./hs.out
"hoge"
```

Please read “Ajhc User’s Manual” to know more detail. (<http://ajhc.metasepi.org/manual.html>)

## Future plans

Fix many BUGs. Try to rewrite (snatch) NetBSD kernel driver with Haskell. If we have luck, will port some library like array or vector from GHC world. After that, we are going to report back about developing Ajhc.

## License

GPL2 or later.

## Contact

- o Mailing list: <http://groups.google.com/group/metasepi>
- o Bug tracker: <https://github.com/ajhc/ajhc/issues>
- o Metasepi team: <https://github.com/ajhc?tab=members>

## Further reading

- o Ajhc – Haskell everywhere: <http://ajhc.metasepi.org/>
- o jhc: <http://repetae.net/computer/jhc/>
- o Metasepi: Project <http://metasepi.org/>
- o Snatch-driven-development: [http://www.slideshare.net/master\\_q/20131020-osc-tokyoajhc](http://www.slideshare.net/master_q/20131020-osc-tokyoajhc)

## 3.3 UHC, Utrecht Haskell Compiler

Report by:	Atze Dijkstra
Participants:	many others
Status:	active development

UHC is the Utrecht Haskell Compiler, supporting almost all Haskell98 features and most of Haskell2010,

plus experimental extensions.

**Status** Current work is on a strictness analyser (Augusto Pasalaqua) and incrementality of analysis via the Attribute Grammar system used to construct UHC (Jeroen Bransen).

**Background.** UHC actually is a series of compilers of which the last is UHC, plus infrastructure for facilitating experimentation and extension. The distinguishing features for dealing with the complexity of the compiler and for experimentation are (1) its stepwise organisation as a series of increasingly more complex standalone compilers, the use of DSL and tools for its (2) aspectwise organisation (called Shuffle) and (3) tree-oriented programming (Attribute Grammars, by way of the Utrecht University Attribute Grammar (UUAG) system (→ 5.3.2).

## Further reading

- o UHC Homepage: <http://www.cs.uu.nl/wiki/UHC/WebHome>
- o UHC Github repository: <https://github.com/UU-ComputerScience/uhc>
- o UHC Javascript backend: <http://uu-computerscience.github.com/uhc-js/>
- o Attribute grammar system: <http://www.cs.uu.nl/wiki/HUT/AttributeGrammarSystem>

## 3.4 Specific Platforms

### 3.4.1 Haskell on FreeBSD

Report by:	PÁLI Gábor János
Participants:	FreeBSD Haskell Team
Status:	ongoing

The FreeBSD Haskell Team is a small group of contributors who maintain Haskell software on all actively supported versions of FreeBSD. The primarily supported implementation is the Glasgow Haskell Compiler together with Haskell Cabal, although one may also find Hugs and NHC98 in the ports tree. FreeBSD is a Tier-1 platform for GHC (on both i386 and amd64) starting from GHC 6.12.1, hence one can always download vanilla binary distributions for each recent release.

We have a developer repository for Haskell ports that features around 486 ports of many popular Cabal packages. The updates committed to this repository are continuously integrated to the official ports tree on a regular basis. However, the FreeBSD Ports Collection already includes many popular and important Haskell software: GHC 7.6.3, Haskell Platform 2013.2.0.0, Gtk2Hs, wxHaskell, XMonad, Pandoc, Gitit, Yesod, Happstack, Snap, Agda, git-annex, and so

on – all of them have been incorporated into the recent 9.2-RELEASE.

In cooperation with fellow developers, Konstantin Belousov and Dimitry Andric, we have managed to restore the ability to build GHC on 32-bit 10.x FreeBSD systems, so now it is ready to be included in the upcoming 10.0-RELEASE. In addition, it turned out that this bug (in thread signal delivery) can also affect the building process for other platforms as well, which explains some of the strange build breakages our users might have experienced in the past.

If you find yourself interested in helping us or simply want to use the latest versions of Haskell programs on FreeBSD, check out our page at the FreeBSD wiki (see below) where you can find all important pointers and information required for use, contact, or contribution.

### Further reading

<http://wiki.FreeBSD.org/Haskell>

#### 3.4.2 Debian Haskell Group

Report by:	Joachim Breitner
Status:	working

The Debian Haskell Group aims to provide an optimal Haskell experience to users of the Debian GNU/Linux distribution and derived distributions such as Ubuntu. We try to follow the Haskell Platform versions for the core package and package a wide range of other useful libraries and programs. At the time of writing, we maintain 628 source packages.

A system of virtual package names and dependencies, based on the ABI hashes, guarantees that a system upgrade will leave all installed libraries usable. Most libraries are also optionally available with profiling enabled and the documentation packages register with the system-wide index.

The recently released stable Debian release (“wheezy”) provides the Haskell Platform 2012.3.0.0 and GHC 7.4.1, while in Debian unstable, we provide the platform 2013.2.0.0 and GHC 7.6.3.

Debian users benefit from the Haskell ecosystem on 13 architecture/kernel combinations, including the non-Linux-ports KFreeBSD and Hurd.

### Further reading

<http://wiki.debian.org/Haskell>

#### 3.4.3 Fedora Haskell SIG

Report by:	Jens Petersen
Participants:	Shakthi Kannan, Ricky Elrod, Ben Boeckel, Michel Salim, and others
Status:	ongoing

The Fedora Haskell SIG works to provide good Haskell support in the Fedora Project Linux distribution.

Fedora 19 shipped in July with ghc-7.4.2, haskell-platform-2012.4.0.0, and version updates to many other packages. New packages added since the previous release included cabal-rpm, and a bunch of libraries. Recently the Fedora Haskell Packaging Guidelines have been updated and revised which will make it easier for general package reviewers and give better control to packagers.

Fedora 20 development is now at the Beta milestone with ghc-7.6.3, haskell-platform-2013.2.0.0, and numerous version updates: it is due to ship in December. New packages added during the Fedora 20 cycle include idris and various libraries. In Fedora 20 armv7 is now a Fedora primary architecture, and the inclusion of llvm-3.3 has caused problems since it is not properly supported by ghc-7.6.3. Work on packaging Yesod has slowly proceeded.

At the time of writing we have around 250 Haskell source packages in Fedora. The cabal-rpm packaging tool has improved: it can now resolve C library and pkgconfig build dependencies, word-wrap descriptions, and yum install build dependencies.

Work on Fedora 21 is also under way. I expect we will ship ghc-7.8 rather than waiting for Haskell Platform this time: it will be great to have shared library support beyond intel archs, and the many other enhancements. Static libraries will probably be moved into static subpackages

If you want to help with Fedora Haskell packaging, please join our low-traffic mailing-list and the Freenode #fedora-haskell channel. You can also follow @fedora-haskell for irregular updates.

### Further reading

- Homepage: <http://fedoraproject.org/wiki/SIGs/Haskell>
- Mailing-list: <https://admin.fedoraproject.org/mailman/listinfo/haskell>
- Package list: <https://admin.fedoraproject.org/pkgdb/users/packages/haskell-sig>
- Package changes: <http://git.fedorahosted.org/cgit/haskell-sig.git/tree/packages/diffs/f19-f20.compare>

## 4 Related Languages and Language Design

### 4.1 Agda

Report by:	Andreas Abel
Participants:	Nils Anders Danielsson, Ulf Norell, Makoto Takeyama, Stevan Andjelkovic, Jean-Philippe Bernardy, James Chapman, Dominique Devriese, Peter Divianszki, Fredrik Nordvall Forsberg, Olle Fredriksson, Daniel Gustafsson, Alan Jeffrey, Fredrik Lindblad, Guilhem Moulin, Nicolas Pouillard, Andrés Sicard-Ramírez and many more
Status:	actively developed

Agda is a dependently typed functional programming language (developed using Haskell). A central feature of Agda is inductive families, i.e., GADTs which can be indexed by *values* and not just types. The language also supports coinductive types, parameterized modules, and mixfix operators, and comes with an *interactive* interface—the type checker can assist you in the development of your code.

A lot of work remains in order for Agda to become a full-fledged programming language (good libraries, mature compilers, documentation, etc.), but already in its current state it can provide lots of fun as a platform for experiments in dependently typed programming.

Since the release of Agda 2.3.2 in November 2012 the following has happened in the Agda project and community:

- o Ulf Norell gave a keynote speech at ICFP 2013 on dependently typed programming in Agda.
- o Agda has attracted new users, the traffic on the mailing list (and bug tracker) is increasing.
- o About 100 bugs of Agda 2.3.2 have been fixed; and small enhancements improve the usability.
- o Copatterns are being added to Agda as a new way to define record and coinductive values.

Release of Agda 2.3.4 is planned to happen soon after the one of GHC 7.8.

#### Further reading

The Agda Wiki: <http://wiki.portal.chalmers.se/agda/>

### 4.2 MiniAgda

Report by:	Andreas Abel
Status:	experimental

MiniAgda is a tiny dependently-typed programming language in the style of Agda (→ 4.1). It serves as a laboratory to test potential additions to the language and type system of Agda. MiniAgda’s termination checker is a fusion of sized types and size-change termination and supports coinduction. Bounded size quantification and destructor patterns for a more general handling of coinduction. Equality incorporates eta-expansion at record and singleton types. Function arguments can be declared as static; such arguments are discarded during equality checking and compilation.

Recently, I have added pattern inductive families that *need not store their indices* even at type-checking time.

MiniAgda is available as Haskell source code and compiles with GHC 6.12.x – 7.6.3.

#### Further reading

<http://www2.tcs.ifi.lmu.de/~abel/miniagda/>

### 4.3 Disciple

Report by:	Ben Lippmeier
Status:	experimental, active development

Disciple Core is an explicitly typed language based on System-F2, intended as an intermediate representation for a compiler. In addition to the polymorphism of System-F2 it supports region, effect and closure typing. Evaluation order is left-to-right call-by-value by default, but explicit lazy evaluation is also supported. The language includes a capability system to track whether objects are mutable or constant, and to ensure that computations that perform visible side effects are not suspended with lazy evaluation.

The Disciplined Disciple Compiler (DDC) is being rewritten to use the redesigned Disciple Core language. This new DDC is at a stage where it will parse and type-check core programs, and compile first-order functions over lists to executables via C or LLVM backends. There is also an interpreter that supports the full language.

#### What is new?

- o Over the last month we’ve been working on a new core language fragment, Disciple Core Flow, to support work on array fusion for Data Parallel Haskell (DPH). We’re writing a GHC plugin that translates GHC core programs to Disciple Core Flow, performs array fusion, and translates back. We’re using Disciple Core Flow instead of GHC Core directly because



it has a simple (and working) external core format, which we use to test the fusion transform.

### Further reading

<http://disciple.ouroborus.net>

## 4.4 Ermine

Report by:	Edward Kmett
Participants:	Dan Doel, Josh Cough, Elliot Stern, Stephen Compall, Runar Oli Bjarnason, Paul Chiusano
Status:	actively developed, experimental

Ermine is a Haskell-like programming language, extended with rank-N types, kind and row polymorphism that runs on the JVM designed at McGraw Hill Financial.

The language currently has two implementations, a legacy implementation that was written in Scala, and a newer, more extensible, implementation that is actively being developed in Haskell.

The Scala implementation is designed more or less as a straight interpreter, while the Haskell version is designed to be able to compile down to a smaller, relatively portable core. Neither backend generates Java bytecode directly to avoid leaking “Permgen” space.

In July, we were able to obtain corporate approval to open source the existing Scala-based compiler and the nascent Haskell implementation. The Scala version of the language is being actively used to generate a number of financial reports within the S&P Capital IQ web platform.

An introduction to Ermine has been given at Boston Haskell and at CUF 2013. Stephen Compall has been putting together a documentation project.

### Further reading

- o Ermine Github: <http://github.com/ermine-language>
- o Boston Haskell Presentation: <http://www.youtube.com/watch?v=QCvXIOCB5A>
- o A Taste of Ermine: <https://launchpad.net/ermine-user-guide>
- o CUF Slides: <http://tinyurl.com/qem8phk>

## 5 Haskell and . . .

### 5.1 Haskell and Parallelism

#### 5.1.1 Eden

Report by:	Rita Loogen
Participants:	in <b>Madrid</b> : Yolanda Ortega-Mallén, Mercedes Hidalgo, Lidia Sánchez-Gil, Fernando Rubio, Alberto de la Encina, in <b>Marburg</b> : Mischa Dieterle, Thomas Horstmeyer, Oleg Lobachev, Rita Loogen, in <b>Copenhagen</b> : Jost Berthold
Status:	ongoing

Eden extends Haskell with a small set of syntactic constructs for explicit process specification and creation. While providing enough control to implement parallel algorithms efficiently, it frees the programmer from the tedious task of managing low-level details by introducing automatic communication (via head-strict lazy lists), synchronization, and process handling.

Eden's primitive constructs are process abstractions and process instantiations. The Eden logo



consists of four  $\lambda$  turned in such a way that they form the Eden instantiation operator ( $\#$ ). Higher-level coordination is achieved by defining *skeletons*, ranging from a simple parallel map to sophisticated master-worker schemes. They have been used to parallelize a set of non-trivial programs.

Eden's interface supports a simple definition of arbitrary communication topologies using *Remote Data*. A *PA-monad* enables the *eager* execution of user defined sequences of *Parallel Actions* in Eden.

#### Survey and standard reference

Rita Loogen, Yolanda Ortega-Mallén, and Ricardo Peña: *Parallel Functional Programming in Eden*, Journal of Functional Programming 15(3), 2005, pages 431–475.

#### Tutorial

Rita Loogen: *Eden - Parallel Functional Programming in Haskell*, in: V. Zsóok, Z. Horváth, and R. Plasmeijer (Eds.): CFP 2011, Springer LNCS 7241, 2012, pp. 142–206.

(see also: <http://www.mathematik.uni-marburg.de/~eden/?content=cefp>)

#### Implementation

Eden is implemented by modifications to the Glasgow-Haskell Compiler (extending its runtime system to use multiple communicating instances). Apart from MPI or PVM in cluster environments, Eden supports a shared memory mode on multicore platforms, which uses multiple independent heaps but does not depend on any middleware. Building on this runtime support, the Haskell package *edenmodules* defines the language, and *edenskels* provides a library of parallel skeletons.

A new version based on GHC-7.8.1 (including binary packages and prepared source bundles) will be released as soon as its baseline GHC is released (expected release date: November 25). The new version fixes a number of issues related to error shut-down and recovery, and features extended support for serialising Haskell data structures. The previous stable release with binary packages and bundles was based on GHC 7.4.2, and is still available on the Eden web pages.

The source code repository for Eden releases is <http://james.mathematik.uni-marburg.de:8080/gitweb>, the Eden libraries (Haskell-level) are also available via Hackage.

#### Tools and libraries

The Eden trace viewer tool *EdenTV* provides a visualisation of Eden program runs on various levels. Activity profiles are produced for processing elements (machines), Eden processes and threads. In addition message transfer can be shown between processes and machines. EdenTV is written in Haskell and is freely available on the Eden web pages and on hackage.

The Eden skeleton library is under constant development. Currently it contains various skeletons for parallel maps, workpools, divide-and-conquer, topologies and many more. Take a look on the Eden pages.

#### Recent and Forthcoming Publications

- Mischa Dieterle, Thomas Horstmeyer, Jost Berthold, Rita Loogen: *Iterating Skeletons — Structured Parallelism by Composition*, Selected Papers of the Symposium on the Implementation and Application of Functional Languages (IFL 2012), LNCS 8241, Springer 2013, pp. 18–36.
- M. KH. Aswad, P. W. Trinder, A. D. Al-Zain, G. J. Michaelson, J. Berthold: *Comparing Low-Pain and No-Pain Multicore Haskell*, revised and extended version of TFP 2009 paper, in Special Issue of Higher-Order Symbol Computation (HOSC), to appear.

- Thomas Horstmeyer and Rita Loogen: *Graph-Based Communication in Eden*, revised and extended version of TFP 2009 paper, in Special Issue of Higher-Order Symbol Computation (HOSC), to appear.
- Oleg Lobachev, Michael Guthe, Rita Loogen: *Estimating parallel performance*, Journal of Parallel and Distributed Computing, Vol 73, No. 6, June 2013, pp. 876 - 887.
- Jost Berthold: *Run-time supported Haskell Serialisation - an API*. Talk at the Haskell Implementors' Workshop, Boston, September 2013.

## Further reading

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

### 5.1.2 Parallel GHC project

Report by:	Duncan Coutts
Participants:	Duncan Coutts, Andres Löh, Mikolaj Konarski, Edsko de Vries
Status:	active

Microsoft Research funded a 2-year project, which is now coming to an end, to promote the real-world use of parallel Haskell. The project involved industrial partners working on their own tasks using parallel Haskell, and consulting and engineering support from Well-Typed (→ 8.1). The overall goal has been to demonstrate successful serious use of parallel Haskell, and along the way to apply engineering effort to any problems with the tools that the organisations might run into. In addition we have put significant engineering work into a new implementation of Cloud Haskell.

The participating organisations are working on a diverse set of complex real world problems:

- Dragonfly (New Zealand): Hierarchical Bayesian Modeling
- Los Alamos National Laboratory (USA): high performance Monte Carlo algorithms to model the flow of radiation and other physical phenomena
- IJ Innovation Institute Inc. (Japan): network servers handling a massive number of concurrent connections
- Telefonica I+D: processing large graphs representing social networks

As the project winds down, we will be publishing more details about the outcomes of these projects.

On the engineering side, the two main areas of focus in the project recently have been ThreadScope and Cloud Haskell.

**ThreadScope.** The latest release of ThreadScope (version 0.2.2) provides detailed statistics about heap and GC behaviour. It is much like the output that can be obtained by running your program with `+RTS -s` but presented in a more friendly way and with the ability to see the same statistics for any period within the program, not just the entire program run. This work could be extended to show graphs of the heap size over time. Compared to GHC's traditional heap profiling this does not require recompiling in profiling mode and is very low overhead, but what is lost is the detailed breakdown of the heap by type, cost centre or retainer.

In addition there is a new feature to emit phase markers from user code and have these visualised in the ThreadScope timeline window.

These new features rely on the development version of GHC, and so will become generally available with GHC-7.8.

Finally, there is an alpha release of an ambitious new feature to integrate data from Linux's "perf" system into ThreadScope. The Linux "perf" system lets us see events in the OS such as system calls and other internal kernel trace points, and also to collect detailed CPU performance counters. Our work has focused on capturing and transforming this data source, and integrating it with the existing RTS event tracing system which we believe will enable many useful new visualisations. Our initial new visualisation in ThreadScope lets us see when system calls are occurring. We hope that this and other future work in this area will help developers who are trying to optimise the performance of applications like network servers.

**Cloud Haskell.** For about the last year we have been working on a new implementation of Cloud Haskell. This is the same idea for concurrent distributed programming in Haskell that Simon Peyton Jones has been telling everyone about, but it's a new implementation designed to be robust and flexible.

The summary about the new implementation is that it exists, it works, it's on hackage, and we think it is now ready for serious experiments.

Compared to the previous prototype:

- it is much faster;
- it can run on multiple kinds of network;
- has backends to support different environments (like cluster or cloud);
- has a new system for dealing with node disconnect and reconnect;
- has a more precisely defined semantics;
- supports composable, polymorphic serialisable closures;
- and internally the code is better structured and easier to work with.

By the time you read this, we will have also released a backend for the Windows Azure cloud platform. Backends for other environments should be relatively straightforward to develop.

Further details including papers, videos and blog posts are on the Cloud Haskell homepage.

### Further reading

- Parallel GHC project homepage: [http://www.haskell.org/haskellwiki/Parallel\\_GHC\\_Project](http://www.haskell.org/haskellwiki/Parallel_GHC_Project)
- Cloud Haskell homepage: [http://www.haskell.org/haskellwiki/Cloud\\_Haskell](http://www.haskell.org/haskellwiki/Cloud_Haskell)
- ThreadScope homepage: <http://www.haskell.org/haskellwiki/ThreadScope>

### 5.1.3 speculation

Report by:	Edward Kmett
Participants:	Jake McArthur
Status:	stable

This package provides speculative function application and speculative folds based on

- Prakash Prabhu, G. Ramalingam, and Kapil Vaswani, “Safe Programmable Speculative Parallelism”, In the proceedings of Programming Language Design and Implementation (PLDI) Vol 45, Issue 6 (June 2010) pp 50-61.

Unlike the original paper, we can take advantage of immutability and the spark queue in Haskell to ensure we never worsen the asymptotics of a single-threaded algorithm. Speculative STM transactions take the place of the transactional rollback machinery from the paper.

### Further reading

- <http://hackage.haskell.org/package/speculation>
- <http://research.microsoft.com/pubs/118795/pldi026-vaswani.pdf>

## 5.2 Haskell and the Web

### 5.2.1 WAI

Report by:	Michael Snoyman
Participants:	Greg Weber
Status:	stable

The Web Application Interface (WAI) is an interface between Haskell web applications and Haskell web servers. By targeting the WAI, a web framework or web application gets access to multiple deployment platforms. Platforms in use include CGI, the Warp web server, and desktop webkit.

WAI has mostly been stable since the last HCAR, with the exception of a newly added field to represent the request body length. This avoids repeatedly doing

a costly integer parse, and correctly handling the case of chunked bodies at the type level. WAI has also been updated to allow the newest version of the conduit (→ 7.1.1) package.

WAI is also a platform for re-using code between web applications and web frameworks through WAI middleware and WAI applications. WAI middleware can inspect and transform a request, for example by automatically gzipping a response or logging a request. The Yesod (→ 5.2.6) web framework provides the ability to embed arbitrary WAI applications as subsites, making them a part of a larger web application.

By targeting WAI, every web framework can share WAI code instead of wasting effort re-implementing the same functionality. There are also some new web frameworks that take a completely different approach to web development that use WAI, such as web-wire (FRP) and dingo (GUI). The Scotty web framework also continues to be developed, and provides a lighter-weight alternative to Yesod. Other frameworks—whether existing or newcomers—are welcome to take advantage of the existing WAI architecture to focus on the more innovative features of web development.

WAI applications can send a response themselves. For example, `wai-app-static` is used by Yesod to serve static files. However, one does not need to use a web framework, but can simply build a web application using the WAI interface alone. The Hoogle web service targets WAI directly.

The WAI standard has proven itself capable for different users and there are no outstanding plans for changes or improvements.

### Further reading

<http://www.yesodweb.com/book/wai>

### 5.2.2 Warp

Report by:	Michael Snoyman
------------	-----------------

Warp is a high performance, easy to deploy HTTP server backend for WAI (→ 5.2.1). Since the last HCAR, Warp has switched from enumerators to conduits (→ 7.1.1), added SSL support, and websockets integration.

Due to the combined use of ByteStrings, blaze-builder, conduit, and GHC’s improved I/O manager, WAI+Warp has consistently proven to be Haskell’s most performant web deployment option.

Warp is actively used to serve up most of the users of WAI (and Yesod).

“Warp: A Haskell Web Server” by Michael Snoyman was published in the May/June 2011 issue of IEEE Internet Computing:

- Issue page: <http://www.computer.org/portal/web/csdl/abs/mags/ic/2011/03/mic201103toc.htm>
- PDF: [http://steve.vinoski.net/pdf/IC-Warp\\_a\\_Haskell\\_Web\\_Server.pdf](http://steve.vinoski.net/pdf/IC-Warp_a_Haskell_Web_Server.pdf)

### 5.2.3 Holumbus Search Engine Framework

Report by:	Uwe Schmidt
Participants:	Timo B. Kranz, Sebastian Gauck, Stefan Schmidt
Status:	first release

#### Description

The Holumbus framework consists of a set of modules and tools for creating fast, flexible, and highly customizable search engines with Haskell. The framework consists of two main parts. The first part is the indexer for extracting the data of a given type of documents, e.g., documents of a web site, and store it in an appropriate index. The second part is the search engine for querying the index.

An instance of the Holumbus framework is the Haskell API search engine Hayoo! (<http://holumbus.fh-wedel.de/hayoo/>).

The framework supports distributed computations for building indexes and searching indexes. This is done with a MapReduce like framework. The MapReduce framework is independent of the index- and search-components, so it can be used to develop distributed systems with Haskell.

The framework is now separated into four packages, all available on Hackage.

- The Holumbus Search Engine
- The Holumbus Distribution Library
- The Holumbus Storage System
- The Holumbus MapReduce Framework

The search engine package includes the indexer and search modules, the MapReduce package bundles the distributed MapReduce system. This is based on two other packages, which may be useful for their on: The Distributed Library with a message passing communication layer and a distributed storage system.

#### Features

- Highly configurable crawler module for flexible indexing of structured data
- Customizable index structure for an effective search
- *find as you type* search
- Suggestions
- Fuzzy queries
- Customizable result ranking
- Index structure designed for distributed search
- Git repository containing the current development version of all packages under <https://github.com/fortytools/holumbus>
- Distributed building of search indexes

#### Current Work

Currently there are activities to optimize the index structures of the framework. In the past there have been problems with the space requirements during indexing. The data structures and evaluation strategies

have been optimized to prevent space leaks. A second index structure working with cryptographic keys for document identifiers is under construction. This will further simplify partial indexing and merging of indexes.

There is a small project extracting the sources of the data structure used for the index to build a separate package. The search tree used in Holumbus is a space optimised version of a radix tree, which enables fast prefix and fuzzy search.

The second project, a specialized search engine for the FH-Wedel web site, has been finished <http://w3w.fh-wedel.de/>. The new aspect in this application is a specialized free text search for appointments, deadlines, announcements, meetings and other dates.

The Hayoo! and the FH-Wedel search engine have been adopted to run on top of the Snap framework (→ 5.2.7).

#### Further reading

The Holumbus web page (<http://holumbus.fh-wedel.de/>) includes downloads, Git web interface, current status, requirements, and documentation. Timo Kranz's master thesis describing the Holumbus index structure and the search engine is available at <http://holumbus.fh-wedel.de/branches/develop/doc/thesis-searching.pdf>. Sebastian Gauck's thesis dealing with the crawler component is available at <http://holumbus.fh-wedel.de/src/doc/thesis-indexing.pdf>. The thesis of Stefan Schmidt describing the Holumbus MapReduce is available via <http://holumbus.fh-wedel.de/src/doc/thesis-mapreduce.pdf>.

### 5.2.4 Happstack

Report by:	Jeremy Shaw
------------	-------------

Happstack is a fast, modern framework for creating web applications. Happstack is well suited for MVC and RESTful development practices. We aim to leverage the unique characteristics of Haskell to create a highly-scalable, robust, and expressive web framework.

Happstack pioneered type-safe Haskell web programming, with the creation of technologies including web-routes (type-safe URLs) and acid-state (native Haskell database system). We also extended the concepts behind formlets, a type-safe form generation and processing library, to allow the separation of the presentation and validation layers.

Some of Happstack's unique advantages include:

- a large collection of flexible, modular, and well documented libraries which allow the developer to choose the solution that best fits their needs for databases, templating, routing, etc.
- the most flexible and powerful system for defining type-safe URLs.

- a type-safe form generation and validation library which allows the separation of validation and presentation without sacrificing type-safety
- a powerful, compile-time HTML templating system, which allows the use of XML syntax

A recent addition to the Happstack family is the `happstack-foundation` library. It combines what we believe to be the best choices into a nicely integrated solution. `happstack-foundation` uses:

- `happstack-server` for low-level HTTP functionality
- `acid-state` for type-safe database functionality
- `web-routes` for type-safe URL routing
- `reform` for type-safe form generation and processing
- `HSP` for compile-time, XML-based HTML templates
- `JMacro` for compile-time Javascript generation and syntax checking

### Future plans

Happstack is the oldest, actively developed Haskell web framework. We are continually studying and applying new ideas to keep Happstack fresh. By the time the next release is complete, we expect very little of the original code will remain. If you have not looked at Happstack in a while, we encourage you to come take a fresh look at what we have done.

Some of the projects we are currently working on include:

- a fast pipes-based HTTP and websockets backend with a high level of evidence for correctness
- a dynamic plugin loading system
- a more expressive system for weakly typed URL routing combinators
- a new system for processing form data which allows fine grained enforcement of RAM and disk quotas and avoids the use of temporary files
- a major refactoring of `HSP` (fewer packages, migration to `Text/Builder`, a `QuasiQuoter`, and more).

One focus of Happstack development is to create independent libraries that can be easily reused. For example, the core `web-routes` and `reform` libraries are in no way Happstack specific and can be used with other Haskell web frameworks. Additionally, libraries that used to be bundled with Happstack, such as `IxSet`, `SafeCopy`, and `acid-state`, are now independent libraries. The new backend will also be available as an independent library.

When possible, we prefer to contribute to existing libraries rather than reinvent the wheel. For example,

our preferred templating library, `HSP`, was created by and is still maintained by Niklas Broberg. However, a significant portion of `HSP` development in the recent years has been fueled by the Happstack team.

We are also working directly with the Fay team to bring an improved type-safety to client-side web programming. In addition to the new `happstack-fay` integration library, we are also contributing directly to Fay itself.

For more information check out the `happstack.com` website — especially the “Happstack Philosophy” and “Happstack 8 Roadmap”.

### Further reading

- <http://www.happstack.com/>
- <http://www.happstack.com/docs/crashcourse/index.html>

### 5.2.5 Mighttpd2 — Yet another Web Server

Report by:	Kazu Yamamoto
Status:	open source, actively developed

`Mighttpd` (called `mighty`) version 2 is a simple but practical Web server in Haskell. It is now working on `Mew.org` serving static files, CGI (mailman and contents search) and reverse proxy for back-end `Yesod` applications.

`Mighttpd` is based on `Warp` providing performance on par with `nginx`. You can use the `mightyctl` command to reload configuration files dynamically and shutdown `Mighttpd` gracefully.

You can install `Mighttpd 2` (`mighttpd2`) from `HackageDB`.

Michael Snoyman and I are now working on `WAI 2`. So far, we improved the performance of `Warp` much. `Mighty` version 3 will be based on this new `Warp` and also provides much faster logger than the `fast-logger` package.

### Further reading

- <http://www.mew.org/~kazu/proj/mighttpd/en/>
- <http://www.ij.ad.jp/en/company/development/tech/mighttpd/>
- <http://aosabook.org/en/posa/warp.html>

### 5.2.6 Yesod

Report by:	Michael Snoyman
Participants:	Greg Weber, Luite Stegeman, Felipe Lessa
Status:	stable

`Yesod` is a traditional MVC RESTful framework. By applying Haskell’s strengths to this paradigm, `Yesod` helps users create highly scalable web applications.

Performance scalability comes from the amazing GHC compiler and runtime. GHC provides fast code and built-in evented asynchronous IO.

But Yesod is even more focused on scalable development. The key to achieving this is applying Haskell's type-safety to an otherwise traditional MVC REST web framework.

Of course type-safety guarantees against typos or the wrong type in a function. But Yesod cranks this up a notch to guarantee common web application errors won't occur.

- declarative routing with type-safe urls — say goodbye to broken links
- no XSS attacks — form submissions are automatically sanitized
- database safety through the Persistent library (→ 7.6.2) — no SQL injection and queries are always valid
- valid template variables with proper template insertion — variables are known at compile time and treated differently according to their type using the shakesporean templating system.

When type safety conflicts with programmer productivity, Yesod is not afraid to use Haskell's most advanced features of Template Haskell and quasi-quoting to provide easier development for its users. In particular, these are used for declarative routing, declarative schemas, and compile-time templates.

MVC stands for model-view-controller. The preferred library for models is Persistent (→ 7.6.2). Views can be handled by the Shakespeare family of compile-time template languages. This includes Hamlet, which takes the tedium out of HTML. Both of these libraries are optional, and you can use any Haskell alternative. Controllers are invoked through declarative routing and can return different representations of a resource (html, json, etc).

Yesod is broken up into many smaller projects and leverages Wai (→ 5.2.1) to communicate with the server. This means that many of the powerful features of Yesod can be used in different web development stacks that use WAI such as Scotty.

The new 1.2 release of Yesod, introduces a number of simplifications, especially to the subsite handling. Most applications should be able to upgrade easily. Some of the notable features are:

- Much more powerful multi-representation support via the selectRep/provideRep API.
- More efficient session handling.
- All Handler functions live in a typeclass, providing you with auto-lifting.

- Type-based caching of responses via the cached function.
- More sensible subsite handling, switch to HandlerT/WidgetT transformers.
- Simplified dispatch system, including a lighter-weight Yesod.
- Simplified streaming data mechanism, for both database and non-database responses.
- Completely overhauled yesod-test, making it easier to use and providing cleaner integration with hspect.
- yesod-auth's email plugin now supports logging in via username in addition to email address.
- Refactored persistent module structure for clarity and ease-of-use.
- Easy asset combining for static javascript and css files
- Faster shakesporean template reloading and support for TypeScript templates.

Since the 1.0 release, Yesod has maintained a high level of API stability, and we intend to continue this tradition. The 1.2 release introduces a lot of potential code breakage, but most of the required updates should be very straightforward. Future directions for Yesod are now largely driven by community input and patches. We've been making progress on the goal of easier client-side interaction, and have high-level interaction with languages like Fay, TypeScript, and CoffeeScript.

The Yesod site (<http://www.yesodweb.com/>) is a great place for information. It has code examples, screencasts, the Yesod blog and — most importantly — a book on Yesod.

To see an example site with source code available, you can view Haskellers (→ 1.2) source code: (<https://github.com/snoyberg/haskellers>).

### Further reading

<http://www.yesodweb.com/>

### 5.2.7 Snap Framework

Report by:	Doug Beardsley
Participants:	Gregory Collins, Shu-yu Guo, James Sanders, Carl Howells, Shane O'Brien, Ozgun Ataman, Chris Smith, Jurrien Stutterheim, Gabriel Gonzalez, and others
Status:	active development

The Snap Framework is a web application framework built from the ground up for speed, reliability, and ease of use. The project's goal is to be a cohesive high-level

platform for web development that leverages the power and expressiveness of Haskell to make building websites quick and easy.

In the six months since the last HCAR Snap has had two major new releases. In 0.12 we added features enabling more efficient and granular reloading of snap applications, allowing a much faster development cycle. The 0.13 release centered around a new high-level API for Heist’s compiled splices. The new API greatly simplifies compiled Heist. In addition, we introduced a new syntax that makes defining splices more convenient.

Our next major milestone on the roadmap is to put the finishing touches on our complete rewrite of the Snap web server using `io-streams`. If you would like to contribute, stop by the `#snapframework` IRC channel on Freenode to keep up with the latest activity.

### Further reading

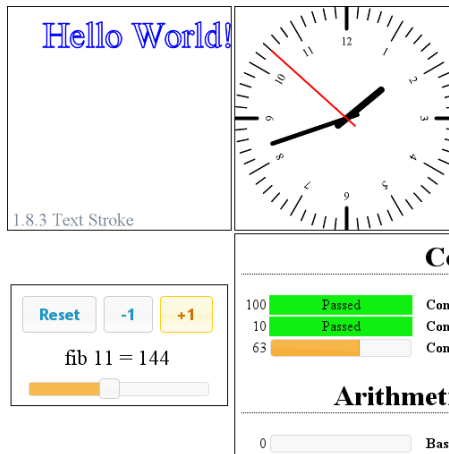
- o 0.12 release announcement: <http://snapframework.com/blog/2013/05/15/snap-0.12-released>
- o 0.13 release announcement: <http://snapframework.com/blog/2013/09/09/snap-0.13-released>
- o A blog post demonstrating the faster reloading capability: <http://devblog.soostone.com/posts/2013-06-17-snap-template-reloading.html>
- o Snaplet Directory: <http://snapframework.com/snaplets>
- o <http://snapframework.com>

### 5.2.8 Sunroof

Report by:	Andy Gill
Participants:	Jan Bracker
Status:	active

Sunroof is a Domain Specific Language (DSL) for generating JavaScript. It is built on top of the JS-monad, which, like the Haskell IO-monad, allows read and write access to external resources, but specifically JavaScript resources. As such, Sunroof is primarily a feature-rich foreign function API to the browser’s JavaScript engine, and all the browser-specific functionality, like HTML-based rendering, event handling, and drawing to the HTML5 canvas.

Furthermore, Sunroof offers two threading models for building on top of JavaScript, atomic and blocking threads. This allows full access to JavaScript APIs, but using Haskell concurrency abstractions, like MVars and Channels. In combination with the push mechanism Kansas-Comet, Sunroof offers a great platform to build interactive web applications, giving the ability to interleave Haskell and JavaScript computations with each other as needed.



It has successfully been used to write smaller applications. These applications range from 2D rendering using the HTML5 canvas element, over small GUIs, up to executing the QuickCheck tests of Sunroof and displaying the results in a neat fashion. The development has been active over the past 6 months and there is a drafted paper submitted to TFP 2013.

### Further reading

- o Homepage: <http://www.ittc.ku.edu/csdl/fpg/software/sunroof.html>
- o Tutorial: <https://github.com/ku-fpg/sunroof-compiler/wiki/Tutorial>
- o Main Repository: <https://github.com/ku-fpg/sunroof-compiler>

### 5.2.9 MFlow

Report by:	Alberto Gómez Corona
Status:	active development

MFlow is an innovative Web framework of the kind of other functional, stateful frameworks like WASH, Seaside, Ocsigen or Racket. MFlow does not use continuation passing, but a backtracking monad that carries out the synchronization of browser request and server process execution state. This monad is on top of another “Workflow” monad that adds effects for logging and recovery of process/session state. Logging/recovery was the continuation mechanism used by WASH, but unlike WASH, MFlow processes don’t die and recover on each request, but stay in memory and synchronize using backtracking. Both effects are managed by an application server that dispatches each request to the appropriate server process. In case of timeout, the process is restarted using the state recovering mechanism.

The logs generated are very small, so MFlow solves the problems of fat serializations associated with stateful frameworks. Like the continuation-based frameworks, MFlow inverts back the inversion of control, so programming multipage web applications is simple and natural. At the same time MFlow is RESTful. Not only the URL use the REST style, but also any GET page



in the flow can be pointed to with a REST URL. That is unique for a stateful WF.

Additionally, besides applicative formlets, it also implements monadic formlets by means of a page monad that permits to dynamically change the page content and input fields depending on the user input. This is called “page flows”. widgets can have their own flow within a page.

If desired, thanks to the runtime templates, the HTML layout and content generated can be modified at runtime. using a web-based WYSIWYG tool included, so the separation of layout and logic is beyond any other web framework.

Other features are widget autorefreshing via implicit AJAX, container widgets (widgets that contain widgets), integration with Persistent, integration with AWS, JQuery integration, WAI integration, blaze-html integration, push using long polling and complete execution traces for errors (thanks to the backtracking monad).

MFlow hides the heterogeneous elements of a web application and expose a clear, modular, type safe DSL of applicative and monadic combinators that pack the client side javascript, the HTML rendering, the CSS and the server code to create from multipage applications to single page, dynamic interfaces.

It may be considered the continuation of the great WASH framework.

### Further reading

There is a site, made using MFlow, which includes demos at: <http://mflowdemo.herokuapp.com>

## 5.3 Haskell and Compiler Writing

### 5.3.1 MateVM

Report by:	Bernhard Urban
Participants:	Harald Steinlechner
Status:	active development

MateVM is a method-based Java Just-In-Time Compiler. That is, it compiles a method to native code on demand (i.e. on the first invocation of a method). We use existing libraries:

**hs-java** for processing Java Classfiles according to *The Java Virtual Machine Specification*.

**harpy** enables runtime code generation for i686 machines in Haskell, in a domain specific language style.

We think that Haskell is suitable for compiler challenges, as already many times proven. However, we have to jump between “Haskell world” and “native code world”, due to the requirements of a Just-In-Time Compiler. This poses some special challenges when it comes to signal handling and other interesting rather

low level operations. Not immediately visible, the task turns out to be well suited for Haskell although we experienced some tensions with signal handling and GHCi. We are looking forward to sharing our experience on this.

In the current state we are able to execute simple Java programs. The compiler eliminates the JVM stack via abstract interpretation, does a liveness analysis, linear scan register allocation and finally code emission. The architecture enables easy addition of further optimization passes on an intermediate representation.

Future plans are, to add an interpreter to gather profile information for the compiler and also do more aggressive optimizations (e.g. method inlining or stack allocation), using the interpreter as fallback path via deoptimization if a assumption is violated.

Apart from that, many features are missing for a full JVM, most notable are the concept of Classloaders, Floating Point or Threads. We would like to use GNU Classpath as base library some day. Other hot topics are Hoopl and Garbage Collection.

If you are interested in this project, do not hesitate to join us on IRC (#MateVM @ OFTC) or contact us on Github.

### Further reading

- o <https://github.com/MateVM>
- o <http://docs.oracle.com/javase/specs/jvms/se7/html/>
- o <http://hackage.haskell.org/package/hs-java>
- o <http://hackage.haskell.org/package/harpy>
- o <http://www.gnu.org/software/classpath/>
- o <http://hackage.haskell.org/package/hoopl-3.8.7.4>
- o <http://en.wikipedia.org/wiki/Club-Mate>

### 5.3.2 UUAG

Report by:	Jeroen Bransen
Participants:	ST Group of Utrecht University
Status:	stable, maintained

UUAG is the *Utrecht University Attribute Grammar* system. It is a preprocessor for Haskell that makes it easy to write *catamorphisms*, i.e., functions that do to any data type what *foldr* does to lists. Tree walks are defined using the intuitive concepts of *inherited* and *synthesized attributes*, while keeping the full expressive power of Haskell. The generated tree walks are *efficient* in both space and time.

An AG program is a collection of rules, which are pure Haskell functions between attributes. Idiomatic tree computations are neatly expressed in terms of copy, default, and collection rules. Attributes themselves can masquerade as subtrees and be analyzed accordingly (higher-order attribute). The order in which to visit the tree is derived automatically from the attribute computations. The tree walk is a single traversal from the perspective of the programmer.

Nonterminals (data types), productions (data constructors), attributes, and rules for attributes can be specified separately, and are woven and ordered automatically. These aspect-oriented programming features make AGs convenient to use in large projects.

The system is in use by a variety of large and small projects, such as the Utrecht Haskell Compiler UHC ( $\rightarrow$  3.3), the editor Proxima for structured documents (<http://www.haskell.org/communities/05-2010/html/report.html#sect6.4.5>), the Helium compiler (<http://www.haskell.org/communities/05-2009/html/report.html#sect2.3>), the Generic Haskell compiler, UUAG itself, and many master student projects. The current version is 0.9.50 (August 2013), is extensively tested, and is available on Hackage. There is also a Cabal plugin for easy use of AG files in Haskell projects.

Some of the recent changes to the UUAG system are:

**OCaml and Clean support.** We have added OCaml and Clean code generation such that UUAG can also be used in OCaml projects and in Clean projects.

We are currently working on the following enhancements:

**Evaluation scheduling and fixed point computation.**

We are running a project to improve the scheduling algorithms in combination with fixed point computations. The currently implemented algorithms for scheduling AG computations do not fully satisfy our needs; the code we write goes beyond the class of OAGs, but the algorithm by Kennedy and Warren (1976) results in an undesired increase of generated code due to non-linear evaluation orders. However, because we know that our code belongs to the class of linear orderable AGs, we would like to find and algorithm that can find this linear order, and thus lies in between the two existing approaches.

Fixed point computations and ordered AG computations do not easily mix. In some cases however we would like to include fixed point computations inside an ordered AG, so the second aim of this project is to introduce way of letting the user specify such computations, and evaluate these in the right way.

**Incremental evaluation.** We are currently also running a Ph.D. project that investigates incremental evaluation of AGs. In this ongoing work we hope to improve the UUAG compiler by adding support for incremental evaluation, for example by statically generating different evaluation orders based on changes in the input.

**Further reading**

- o <http://www.cs.uu.nl/wiki/bin/view/HUT/AttributeGrammarSystem>
- o <http://hackage.haskell.org/package/uuagc>

### 5.3.3 LQPL — A Quantum Programming Language Compiler and Emulator

Report by: Brett G. Giles  
 Participants: Dr. J.R.B. Cockett  
 Status: v 0.9.1 experimental released in November 2013

LQPL (Linear Quantum Programming Language) is a functional quantum programming language inspired by Peter Selinger’s paper “Towards a Quantum Programming Language”.

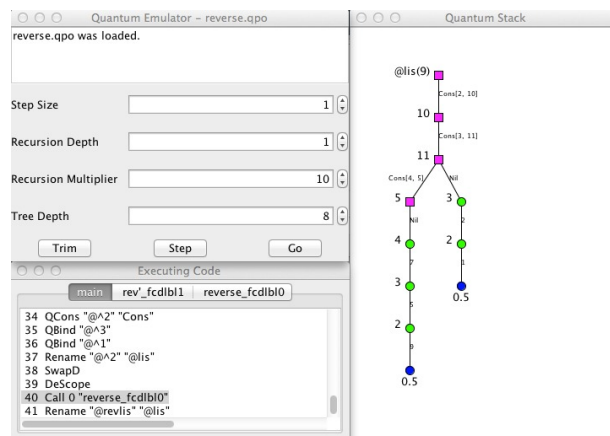
The LQPL system consists of a compiler, a GUI based front end and an emulator. LQPL incorporates a simple module / include system (more like C’s include than Haskell’s import), predefined unitary transforms, quantum control and classical control, algebraic data types, and operations on purely classical data.

Starting with the 0.9 series, LQPL is now split into separate components:

- o The compiler (Haskell) — available at the command line and via a TCP/IP interface;
- o The emulator (which emulates a virtual quantum machine) (Haskell) — available as a server via a TCP/IP interface;
- o The front end (JRuby/Swing) — which connects to both the compiler and the emulator via TCP/IP.

Version 0.9.1 was a bugfix release.

A screenshot of the interface (showing a probabilistic list) is included below.



Quantum programming allows us to provide a fair coin toss:

```

qdata Coin = {Heads | Tails}
toss :: ( ; c:Coin) =
{ q = |0>; Had q;
  measure q of |0> => {c = Heads}
                |1> => {c = Tails}
}

```

This allows programming of probabilistic algorithms, such as leader election.

The next major items on the road map are:

- Change the TCP/IP data format to something less verbose;
- Implementing a translation of the virtual machine code into quantum circuits.

### Further reading

Documentation and executable downloads may be found at <http://pll.cpsc.ucalgary.ca/lqpl/index.html>. The source code, along with a wiki and bug tracker, is available at <https://bitbucket.org/BrettGilesUofC/lqpl>.

### 5.3.4 free — Free Monads

Report by:	Edward Kmett
Participants:	Gabriel Gonzalez, Aristid Breitkreuz, Nickolay Kudasov, Ben Gamari, Matvey Aksenov, Mihaly Barasz, Twan van Laarhoven
Status:	actively developed

This package provides common definitions for working with free monads and free applicatives. These are very useful when it comes to defining EDSLs.

This package also supports cofree comonads, which are useful for tracking attributes through a syntax tree.

Recently support was added for the free completely-iterative monad of a monad as well. This can be used as part of a scheme to deamortize calculations in the *ST s* monad.

### Further reading

- <http://hackage.haskell.org/package/free>
- <http://www.haskellforall.com/2012/06/you-could-have-invented-free-monads.html>
- <http://www.iai.uni-bonn.de/~jv/mpc08.pdf>
- <http://comonad.com/reader/2011/free-monads-for-less/>
- <http://comonad.com/reader/2011/free-monads-for-less-2/>
- <http://comonad.com/reader/2011/free-monads-for-less-3/>
- <http://paolocapriotti.com/assets/applicative.pdf>
- <http://skillsmatter.com/podcast/scala/monads-for-free>
- <http://comonad.com/reader/2009/incremental-folds/>
- <http://www.ioc.ee/~tarmo/tday-veskisilla/uustalu-slides.pdf>
- <https://www.fpcomplete.com/user/edwardk/oblivious/deamortized-st>

### 5.3.5 bound — Making De Bruijn Succ Less

Report by:	Edward Kmett
Participants:	Nicolas Pouillard, Jean-Philippe Bernardy, Andrea Vezzosi, Gabor Greif, Matvey B. Aksenov
Status:	actively developed

This library provides convenient combinators for working with “locally-nameless” terms. These can be useful when writing a type checker, evaluator, parser, or pretty printer for terms that contain binders like forall or lambda, as they ease the task of avoiding variable capture and testing for alpha-equivalence.

Notably, it uses a representation based on type-safe generalized De Bruijn indices that lets you naturally make your expression type into a Monad that permits capture-avoiding substitution, and the use of Foldable’s *toList* and Traversable’s *traverse* to find free variables. This makes it much easier to manipulate your syntax tree with tools you already know how to use, while still safely avoiding issues with name capture.

The generalized De Bruijn encoding permits asymptotic improvement in the running time of many calculations, enabling simultaneous substitution of everything within a complex binder,  $O(1)$  lifting, and avoiding paying for the traversal of lifted trees, but the complexity of the encoding is hidden behind a monad transformer that provides you with variable capture.

### Further reading

- <http://fpcomplete.com/user/edwardk/bound>
- <http://hackage.haskell.org/package/bound>
- <http://www.slideshare.net/ekmett/bound-making-de-bruijn-succ-less>

## 6 Development Tools

### 6.1 Environments

#### 6.1.1 Haskell IDE From FP Complete

Report by: Michael Snoyman  
Status: available, stable

This past September, FP Complete™ officially launched the world's first commercial Haskell IDE and Development Platform, FP Haskell Center™. The IDE includes a Haskell compiler and a continually updated set of vetted, tested and supported libraries and code templates. There is no need to run Cabal or other installers to use it.

Key features include:

- Continuous compile with real-time type information and error messaging
- Haddocks documentation and Hoogle search
- Syntax highlighting
- Git/GitHub integration
- Team management tools
- Remote version control
- An integrated web deployment platform
- Subscriptions include continuous refresh releases on new features, updates, bug fixes and free community support

FP Complete started this project by researching existing solutions to determine what to include in the IDE. They looked at Haskell editor extensions, especially Vim and Emacs, and at existing Haskell IDEs like Eclipse FP, Leksah, and Yi as well as looking at what supporting functionality was readily available: the GHC API, Hoogle, Haddock, Hlint, and many others. From there, FP Complete focused on creating an intuitive interface to make programming within the IDE a seamless experience. To ensure FP Haskell Center is meeting the demands of the Haskell community, FP complete is constantly seeking feedback and suggestions from users and the Haskell community.

Based on the Beta period and the two months the IDE has officially been on the market, the feedback and activity on the IDE has been very positive. One Beta user, Armando Blancas, a Tools Engineer at Carrier IQ, Inc., commented, *“My first impression of FP Haskell IDE is it has a well-designed and polished user interface that feels very intuitive and easy to explore. As a user,*

*I like having things done for me. For example within the IDE, the platform is just there, auto-save, auto-compile, and a push-button program execution lets me just get on with my work.”*

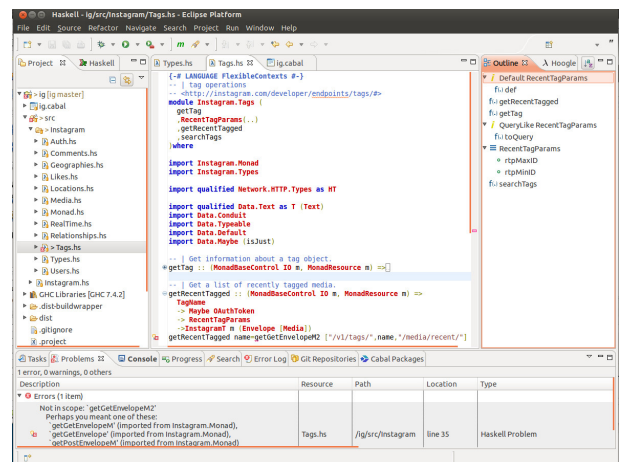
Three levels of subscription are available. The first is the commercial Haskell platform, FP Haskell Center – Professional. The second is FP Haskell Center – Personal, which is designed for non-commercial programmers. Lastly, there is a free academic version for all current students and professors.

#### Further reading

<http://www.fpcomplete.com/>

#### 6.1.2 EclipseFP

Report by: JP Moresmau  
Participants: building on code from B. Scott Michel, Alejandro Serrano, Thiago Arrais, Leif Frenzel, Thomas ten Cate, Martijn Schrage, Adam Foltzer and others  
Status: stable, maintained, and actively developed



EclipseFP is a set of Eclipse plugins to allow working on Haskell code projects. Its goal is to offer a fully featured Haskell IDE in a platform developers coming from other languages may already be familiar with. It provides the following features, among others:

#### Cabal Integration

Provides a .cabal file editor, uses Cabal settings for compilation, allows the user to install Cabal packages from within the IDE. Can also use cabal-dev to provide sandboxing and project dependencies inside an Eclipse workspace.

## GHC Integration

Compilation is done via the GHC API, syntax coloring uses the GHC Lexer.

## Productive Coding

Quick fixes for common errors, warnings, and HLint suggestions. Automatic organization of imports. Autocompletion. Find and rename across modules and projects. Stylish-haskell integration for consistent code formatting.

## Debugging

Easy to launch GHCi sessions on any module with proper parameters. Manages breakpoints, the evaluation of variables and expressions uses the Eclipse debugging framework, and requires no knowledge of GHCi syntax. Also integrates with Yesod (launch the web application from EclipseFP). Running a program with profiling options results in profiling graphs being displayed in the UI for easy analysis.

## Browsing

The Haskell Browser perspective allows the user to navigate the list of packages and their documentation. It integrates seamlessly with Hackage. The Haskell module editor provides code folding, outline view of the module, popup of types and documentation mouse hovers, etc.

## Testing

EclipseFP integrates with Haskell test frameworks, most notably HTF, to provide UI feedback on test failures.

The source code is fully open source (Eclipse License) on github and anyone can contribute. Current version is 2.5.5, released in October 2013, and more versions with additional features are planned and actively worked on. Most notably, version 2.6 should include support for Cabal 1.8 sandboxes. Feedback on what is needed is welcome! The website has information on downloading binary releases and getting a copy of the source code. Support and bug tracking is handled through Sourceforge forums and github issues.

## Further reading

- <http://eclipsefp.github.com/>
- <http://jpmoresmau.blogspot.com/>

### 6.1.3 Ariadne

Report by:	Roman Cheplyaka
Status:	early development

Ariadne is a new tool to provide the go-to-definition functionality for Haskell code. It is designed as a server which responds to queries from IDEs and text editor plugins.

So far there's only a plugin for vim, but we are looking forward to more editors and IDEs getting Ariadne support.

The server uses the `haskell-names` name resolution library (→ 7.3.6). So it is fully Haskell-aware and can properly locate prefixed names (such as `T.head`) and locally bound names, which makes it much smarter than TAGS.

Some of the future work directions are:

- finding definitions in other files (as of 0.1.2, Ariadne can only find definitions in the current file)
- getting information from `.cabal` files (such as the language extensions and dependency information)
- working with unsaved buffers

## Further reading

- Ariadne homepage:  
<https://github.com/feuerbach/ariadne>
- Video demonstration: <http://youtu.be/-sbGijbhxAc>

### 6.1.4 ghc-mod — Happy Haskell Programming

Report by:	Kazu Yamamoto
Status:	open source, actively developed

`ghc-mod` is a backend command to enrich Haskell programming on editors including Emacs, Vim and Sublime. The `ghc-mod` package on Hackage includes the `ghc-mod` command and Emacs front-end.

Emacs front-end provides the following features:

**Completion** You can complete a name of keyword, module, class, function, types, language extensions, etc.

**Code template** You can insert a code template according to the position of the cursor. For instance, “module Foo where” is inserted in the beginning of a buffer.

**Syntax check** Code lines with error messages are automatically highlighted thanks to flymake. You can display the error message of the current line in another window. `hlint` can be used instead of GHC to check Haskell syntax.

**Document browsing** You can browse the module document of the current line either locally or on Hackage.

**Expression type** You can display the type/information of the expression on the cursor.

There are two Vim plugins:

- `ghcmod-vim`
- `syntastic`

Here are new features:

- `ghc-mod` now supports the sandbox of `cabal 1.18` and stopped supporting the sandbox of `cabal-dev`.
- The `ghc-mod` package also provides a library now.

### Further reading

<http://www.mew.org/~kazu/proj/ghc-mod/en/>

### 6.1.5 HEAT: The Haskell Educational Advancement Tool

Report by:	Olaf Chitil
Status:	active

Heat is an interactive development environment (IDE) for learning and teaching Haskell. Heat was designed for novice students learning the functional programming language Haskell. Heat provides a small number of supporting features and is easy to use. Heat is distributed as a single, portable Java jar-file and works on top of GHCi.

Version 5.05, with small improvements and bug-fixes, was released end of April 2013.

Heat provides the following features:

- Editor for a single module with syntax-highlighting and matching brackets.
- Shows the status of compilation: non-compiled; compiled with or without error.
- Interpreter console that highlights the prompt and error messages.
- If compilation yields an error, then the relevant source line is highlighted and no further expression can be evaluated in the console until the source has been changed and successfully recompiled.
- A tree structure provides a program summary, giving definitions of types and types of functions.
- Automatic checking of either Boolean or QuickCheck properties of a program; results shown in summary.

### Further reading

<http://www.cs.kent.ac.uk/projects/heat/>

### 6.1.6 HaRe — The Haskell Refactorer

Report by:	Alan Zimmerman
Participants:	Francisco Soares, Chris Brown

Refactorings are source-to-source program transformations which change program structure and organization, but not program functionality. Documented in catalogs and supported by tools, refactoring provides the means to adapt and improve the design of existing

code, and has thus enabled the trend towards modern agile software development processes.

Our project, *Refactoring Functional Programs*, has as its major goal to build a tool to support refactorings in Haskell. The HaRe tool is now in its seventh major release. HaRe supports full Haskell 2010, and is integrated with (X)Emacs. All the refactorings that HaRe supports, including renaming, scope change, generalization and a number of others, are *module-aware*, so that a change will be reflected in all the modules in a project, rather than just in the module where the change is initiated.

Snapshots of HaRe are available from our GitHub repository (see below) and Hackage. There are related presentations and publications from the group (including LDTA'05, TFP'05, SCAM'06, PEPM'08, PEPM'10, TFP'10, Huiqing's PhD thesis and Chris's PhD thesis). The final report for the project appears on the University of Kent Refactoring Functional Programs page (see below).

There is also a Google+ community called HaRe, and an IRC channel on freenode called `#haskell-refactorer`.

### Recent developments

- HaRe 0.7, which is a major change from 0.6 as it makes use of the GHC library for analysis, has been released; HaRe 0.7 is available on Hackage, and also downloadable from our GitHub page
- HaRe 0.7 is alpha software, and comes with a limited number of refactorings, as the work so far has concentrated on getting the new architecture in place to make use of the GHC AST.
- There is plenty to do, so anyone who has an interest is welcome to fork the repo and get stuck in.
- Stephen Adams is starting his PhD at the University of Kent and will be working on data refactoring in Haskell.

### Further reading

- <http://www.cs.kent.ac.uk/projects/refactor-fp/>
- <https://github.com/alanz/HaRe>

### 6.1.7 IHaskell: Haskell for Interactive Computing

Report by:	Andrew Gibiansky
Status:	very alpha (but usable!)

IHaskell is an interactive interface for Haskell development, designed with the goal of replacing GHCi in some contexts. In addition to a simple REPL, it provides a *notebook* interface (in the style of Mathematica or Maple). The notebook interface runs in a browser and provides the user with editable cells in which they can create and execute code. The output of this code is displayed in a rich format right below, and if it's not quite right, the user can go back, edit the cell, and

re-execute. This rich format defaults to the same boring plain-text output as GHCi would give you; however, library authors will be able to define their own formats for displaying their data structures in a useful way, with the only limit being that the display output must be viewable in a browser (images, HTML, CSS, Javascript). For instance, integration with graphing libraries could produce in-browser data visualizations, while integration with Aeson’s JSON could produce a syntax-highlighted JSON output for complex data structures.

```
In [1]: data Value = X Int
          | Y String
          | Z Float
          deriving Show

let values = [X 20,
              Y "Test",
              Z 0.5]
mapM_ print values
```

```
Out[1]: X 20
        Y "Test"
        Z 0.5
```

```
In [2]: import Control.Applicative

print $ (+) <$> Just 3 <*> Just 10|
```

```
Out[2]: Just 13
```

```
In [4]: import Control.Monad
forM [1, 2, 3, 4] $ \x -> do
  print (x * x)
  return (-x)
```

```
Out[4]: 1
        4
        9
        16
        [-1,-2,-3,-4]
```

Implementation-wise, IHaskell is a language kernel backend for the project known as IPython. Although it has the name “Python” in the name, IPython provides a *language-agnostic* protocol by which interactive code environments such as REPLs and notebooks can communicate with a language evaluator backend. IHaskell is a language kernel which uses ZeroMQ to communicate with IPython frontends and the GHC API to evaluate code.

```
IPython profile: haskell

In [1]: putStrLn "IHaskell!"
Out[1]:
IHaskell!

In [2]: Just 15
Out[2]:
Just 15

In [3]: import Control.Applicative
Out[3]:

In [4]: print $ (*) <$> Just 10 <*> Just 0.5
Out[4]:
Just 5.0
```

Although IHaskell is in very early stages, the future looks incredibly bright. Integration with popular Haskell libraries can give us beautiful and potentially interactive visualizations of Haskell data structures. On one hand, this could range from simple things such as foldable record structures — imagine being able to explore complex nested records by folding and unfolding bits and pieces at a time, instead of trying to mentally parse them from the GHCi output. On the other end, we could have interactive outputs, such as Parsec parsers which generate small input boxes that run the parser on any input they’re given. And these things are just the beginning — tight integration with IPython may eventually be able to provide things such as code-folding in your REPL or an integrated debugger interface.

If this sounds good to you: **contribute!** We’re in dire need of developers to make this beautiful dream a reality, and I would be happy to help you get up to speed quickly.

### Further reading

<https://github.com/gibiansky/IHaskell>

## 6.2 Code Management

### 6.2.1 Darcs

Report by:	Eric Kow
Participants:	darcs-users list
Status:	active development

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

Our most recent release, Darcs 2.8.4 (with GHC 7.6 support), was in February 2013. Some key changes in Darcs 2.8 include a faster and more readable `darcs annotate`, a `darcs obliterate -0` which can be used to conveniently “stash” patches, and hunk editing for the `darcs revert` command.

Our work on the next Darcs release continues. In our sights are the new ‘darcs rebase’ command (for merging and amending patches that would be hard to do with patch theory alone), the patch index optimisation (for faster local lookups on repositories with long histories), and the packs optimisation (for faster darcs get).

This summer we have been lucky to get a chance to work on not just one, but *two* Google Summer of Code projects. (We participate in the Summer of Code under the Haskell.org umbrella, with a special agreement with Google to have an extra slot allocated for the purposes of Darcs work. This year, one of the Darcs related submissions was also selected in addition to the Darcs slot.) Our projects this year were on improving the local patch recording experience (Jose Neder), and the Darcs hub (BSRK Aditya).

For the work on patch recording, Jose Neder brought three major improvements to the Darcs user interface: the patience diff algorithm, making patches much more reflective of human intention; automatic detection of file-move operations, and automatic detection of token replaces, resulting in patches that take better advantage of Darcs' high-level representation of changes.

As for the Darcs hub project (the Darcsden instance running on <http://hub.darcs.net>), BSRK Aditya has brought us a series of now live changes that make the hub easier to get up and running with. OAuth support means that users can now log in with their GitHub or Google accounts avoiding the need to create yet-another-password. A handful of nice new features make the site more useful than just a repository store: file edit for convenient quick edits, repository compare and patch bundle support for enhanced collaboration with forks, changes/annotate on files for better insight on the evolution of you code. Finally, the new `build packs` feature (see repository settings) enables an optimisation (GSoC 2010) which we hope will pave the way to much faster `darcs get` over networks.

These changes have been merged into the mainline Darcs and Darcsden repositories, and are expected to be available in the next stable release of Darcs.

Darcs is free software licensed under the GNU GPL (version 2 or greater). Darcs is a proud member of the Software Freedom Conservancy, a US tax-exempt 501(c)(3) organization. We accept donations at <http://darcs.net/donations.html>.

#### Further reading

- <http://darcs.net>
- <http://darcs.net/Development/Priorities>
- Post GSoC 2013 hub.darcs.net update <http://joyful.com/blog/2013-09-26-darcsden-darcs-hub-gsoc-complete.html>

#### 6.2.2 DarcsWatch

Report by:	Joachim Breitner
Status:	working

DarcsWatch is a tool to track the state of Darcs (→ 6.2.1) patches that have been submitted to some project, usually by using the `darcs send` command. It allows both submitters and project maintainers to

get an overview of patches that have been submitted but not yet applied.

DarcsWatch continues to be used by the xmonad project, the Darcs project itself, and a few developers. At the time of writing (November 2013), it was tracking 39 repositories and 4602 patches submitted by 248 users.

#### Further reading

- <http://darcswatch.nomeata.de/>
- <http://darcs.nomeata.de/darcswatch/documentation.html>

#### 6.2.3 cab — A Maintenance Command of Haskell Cabal Packages

Report by:	Kazu Yamamoto
Status:	open source, actively developed

`cab` is a MacPorts-like maintenance command of Haskell cabal packages. Some parts of this program are a wrapper to `ghc-pkg` and `cabal`.

If you are always confused due to inconsistency of `ghc-pkg` and `cabal`, or if you want a way to check all outdated packages, or if you want a way to remove outdated packages recursively, this command helps you.

`cab` now supports the sandbox of `cabal 1.18` and stopped supporting `cabal-dev`. Some new command-line options are added, including `-j` for parallel jobs, `-p` for library profiling and `-e` for executable profiling.

#### Further reading

<http://www.mew.org/~kazu/proj/cab/en/>

### 6.3 Deployment

#### 6.3.1 Cabal and Hackage

Report by:	Duncan Coutts
------------	---------------

#### Background

Cabal is the standard packaging system for Haskell software. It specifies a standard way in which Haskell libraries and applications can be packaged so that it is easy for consumers to use them, or re-package them, regardless of the Haskell implementation or installation platform.

Hackage is a distribution point for Cabal packages. It is an online archive of Cabal packages which can be used via the website and client-side software such as `cabal-install`. Hackage enables users to find, browse and download Cabal packages, plus view their API documentation.

`cabal-install` is the command line interface for the Cabal and Hackage system. It provides a command line



program `cabal` which has sub-commands for installing and managing Haskell packages.

## Recent progress

The Cabal packaging system has always faced growing pains. We have been through several cycles where we've faced chronic problems, made major improvements which bought us a year or two's breathing space while package authors and users become ever more ambitious and start to bump up against the limits again. In the last few years we have gone from a situation where 10 dependencies might be considered a lot, to a situation now where the major web frameworks have a 100+ dependencies and we are again facing chronic problems.

The Cabal/Hackage maintainers and contributors have been pursuing a number of projects to address these problems:

The IHG sponsored Well-Typed ( $\rightarrow$  8.1) to work on `cabal-install` resulting in a new package dependency constraint solver. This was incorporated into the `cabal-install-0.14` release in the spring, and which is now in the latest Haskell Platform release. The new dependency solver does a much better job of finding install plans. In addition the `cabal-install` tool now warns when installing new packages would break existing packages, which is a useful partial solution to the problem of breaking packages.

We had two Google Summer of Code projects on Cabal this year, focusing on solutions to other aspects of our current problems. The first is a project by Mikhail Glushenkov (and supervised by Johan Tibell) to incorporate sandboxing into `cabal-install`. In this context sandboxing means that we can have independent sets of installed packages for different projects. This goes a long way towards alleviating the problem of different projects needing incompatible versions of common dependencies. There are several existing tools, most notably `cabal-dev`, that provide some sandboxing facility. Mikhail's project was to take some of the experience from these existing tools (most of which are implemented as wrappers around the `cabal-install` program) and to implement the same general idea, but properly integrated into `cabal-install` itself. We expect the results of this project will be incorporated into a `cabal-install` release within the next few months.

The other Google Summer of Code project this year, by Philipp Schuster (and supervised by Andres Löh), is also aimed at the same problem: that of different packages needing inconsistent versions of the same common dependencies, or equivalently the current problem that installing new packages can break existing installed packages. The solution is to take ideas from the Nix package manager for a persistent non-destructive package store. In particular it lifts an obscure-sounding but critical limitation: that of being able to install multiple instances of the same version of a package, built

against different versions of their dependencies. This is a big long-term project. We have been making steps towards it for several years now. Philipp's project has made another big step, but there's still more work before it is ready to incorporate into `ghc`, `ghc-pkg` and `cabal`.

## Looking forward

Johan Tibell and Bryan O'Sullivan have volunteered as new release managers for Cabal. Bryan moved all the tickets from our previous `trac` instance into `github`, allowing us to move all the code to `github`. Johan managed the latest release and has been helping with managing the inflow of patches. Our hope is that these changes will increase the amount of contributions and give us more maintainer time for reviewing and integrating those contributions. Initial indications are positive. Now is a good time to get involved.

The IHG is currently sponsoring Well-Typed to work on getting the new Hackage server ready for switchover, and helping to make the switchover actually happen. We have recruited a few volunteer administrators for the new site. The remaining work is mundane but important tasks like making sure all the old data can be imported, and making sure the data backup system is comprehensive. Initially the new site will have just a few extra features compared to the old one. Once we get past the deployment hurdle we hope to start getting more contributions for new features. The code is structured so that features can be developed relatively independently, and we intend to follow Cabal and move the code to `github`.

We would like to encourage people considering contributing to take a look at the bug tracker on `github`, take part in discussions on tickets and pull requests, or submit their own. The bug tracker is reasonably well maintained and it should be relatively clear to new contributors what is in need of attention and which tasks are considered relatively easy. For more in-depth discussion there is also the `cabal-devel` mailing list.

## Further reading

- Cabal homepage: <http://www.haskell.org/cabal>
- Hackage package collection: <http://hackage.haskell.org/>
- Bug tracker: <https://github.com/haskell/cabal/>

### 6.3.2 Stackage: the Library Dependency Solution

Report by:	Michael Snoyman
Status:	new

Stackage is a project that began in November 2012 with the mission of making it possible to build stable, vetted sets of packages. The overall goal was to make the Cabal ( $\rightarrow$  6.3.1) experience better. A year into the project it's exciting to share there has been a lot of activity by

FP Complete and greater Haskell community to make Stackage a success. The contributions have been amazing, with a huge number of package authors signing up to contribute and maintain their packages on Stackage.

Through Stackage, FP Complete monitors the package upload logs and signals alerts when a new uploader uploads a package for the first time. For example, if Alice uploads the package *alice – package*, and Bob uploads a new version a few days later, Stackage will get a warning and contact Alice. Internally, FP Complete won't build any new library sets until the problem has been resolved.

Each time FP Complete performs a Stackage build, test suites are automatically run, which has uncovered a number of regressions in upstream packages. These reports are filed upstream as soon as possible, and due to the wonderful nature of the Haskell community, bugs tend to be fixed very quickly. As the FP Haskell Center user base grows, more bug reports are coming from users. FP Complete is working with upstream providers to get these bugs triaged and fixed. As part of our stable library mission, FP Complete has the infrastructure in place to maintain patchsets against older versions of packages, to simplify the task of backporting fixes.

Since Stackage started, notifications for outdated dependencies have drastically decreased. This is a sign the Haskell ecosystem is beginning to stabilize. Previously, it was to be expected that breaking releases would happen on a regular basis, but Stackage is changing that. To date, users have contributed roughly 200 separate packages to the Stackage package collection. When all of the deep dependencies of these packages are included, FP Complete is currently building about 480 total packages in a full Stackage build, or roughly 9% of all Hackage packages. This is great news for anyone using Stackage directly to build a package database, as they gain direct access to a huge number of the most popular Haskell packages, without worries of compatibility.

As part of regular Stackage maintenance, three separate daily Stackage builds are maintained, one using the most recent Haskell Platform, one using the previous release, and one using vanilla GHC 7.4 (no Haskell Platform constraints). Every day, these jobs run, try to compile all of these packages, and run the full set of test suites. The process has found a large number of simple compilation issues, overly restrictive cabal version bounds, and a number of actual bugs. Stackage is living up to its goal of helping maintainers raise the quality of their code.

This is an excellent opportunity for the Haskell community to get involved! If you've written some code that you're actively maintaining, get it in Stackage. You'll be widening the potential audience of users for your code by getting your package into FP Haskell Center, and you'll get some helpful feedback from the au-

tomated builds so that users can more reliably build your code.

### Further reading

<https://www.fpcomplete.com/blog/2013/09/state-of-stackage>

### 6.3.3 standalone-haddock

Report by:	Roman Cheplyaka
Status:	working

`standalone-haddock` solves the problem of publishing haddock documentation on the web.

When you simply run `cabal haddock`, the resulting HTML documentation contains hyperlinks to other packages on your system. As a result, you cannot publish it on the internet (well, you can, but the links will be broken).

`standalone-haddock` takes several packages for which you want to publish documentation. It generates documentation for them with proper links:

- links to identifiers inside this package set are relative
- links to identifiers from external packages lead to hackage

Thus the resulting directory with HTML files is relocatable and publishable.

### Further reading

- README: <http://documentup.com/feuerbach/standalone-haddock>
- An example of the generated documentation: <http://haskell-suite.github.io/docs/>

## 6.4 Others

### 6.4.1 lhs2TeX

Report by:	Andres Löh
Status:	stable, maintained

This tool by Ralf Hinze and Andres Löh is a preprocessor that transforms literate Haskell or Agda code into  $\LaTeX$  documents. The output is highly customizable by means of formatting directives that are interpreted by `lhs2TeX`. Other directives allow the selective inclusion of program fragments, so that multiple versions of a program and/or document can be produced from a common source. The input is parsed using a liberal parser that can interpret many languages with a Haskell-like syntax.

The program is stable and can take on large documents.

The current version is 1.18 and has been released in September 2012. The main change is compatibility with GHC 7.6. Development repository and bug

tracker are on GitHub. There are still plans for a rewrite of lhs2 $\text{\TeX}$  with the goal of cleaning up the internals and making the functionality of lhs2 $\text{\TeX}$  available as a library.

### Further reading

- <http://www.andres-loeh.de/lhs2tex>
- <https://github.com/kosmikus/lhs2tex>

### 6.4.2 ghc-heap-view

Report by:	Joachim Breitner
Participants:	Dennis Felsing
Status:	active development

The library ghc-heap-view provides means to inspect the GHC's heap and analyze the actual layout of Haskell objects in memory. This allows you to investigate memory consumption, sharing and lazy evaluation.

This means that the actual layout of Haskell objects in memory can be analyzed. You can investigate sharing as well as lazy evaluation using ghc-heap-view.

The package also provides the GHCi command `:printHeap`, which is similar to the debuggers' `:print` command but is able to show more closures and their sharing behaviour:

```
> let x = cycle [True, False]
> :printHeap x
_bco
> head x
True
> :printHeap x
let x1 = True : _thunk x1 [False]
in x1
> take 3 x
[True,False,True]
> :printHeap x
let x1 = True : False : x1
in x1
```

The graphical tool ghc-vis ( $\rightarrow$  6.4.3) builds on ghc-heap-view.

### Further reading

- <http://www.joachim-breitner.de/blog/archives/548-ghc-heap-view-Complete-referential-opacity.html>
- <http://www.joachim-breitner.de/blog/archives/580-GHCi-integration-for-GHC.HeapView.html>
- <http://www.joachim-breitner.de/blog/archives/590-Evaluation-State-Assertions-in-Haskell.html>

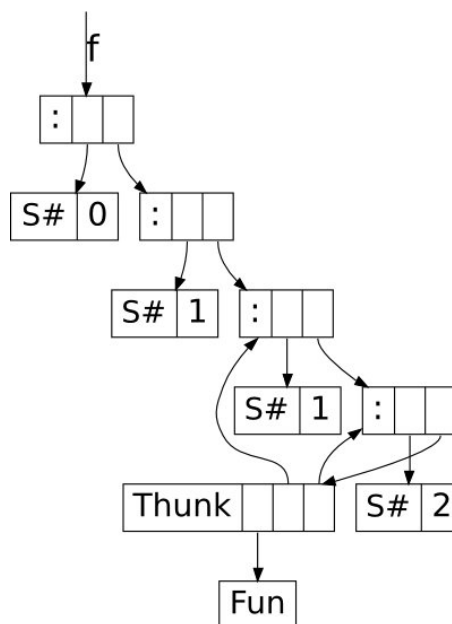
### 6.4.3 ghc-vis

Report by:	Dennis Felsing
Participants:	Joachim Breitner
Status:	active development

The tool ghc-vis visualizes live Haskell data structures in GHCi. Since it does not force the evaluation of the values under inspection it is possible to see Haskell's lazy evaluation and sharing in action while you interact with the data.

Ghc-vis supports two styles: A linear rendering similar to GHCi's `:print`, and a graph-based view where closures in memory are nodes and pointers between them are edges. In the following GHCi session a partially evaluated list of fibonacci numbers is visualized:

```
> let f = 0 : 1 : zipWith (+) f (tail f)
> f !! 2
> :view f
```



At this point the visualization can be used interactively: To evaluate a thunk, simply click on it and immediately see the effects. You can even evaluate thunks which are normally not reachable by regular Haskell code.

Ghc-vis can also be used as a library and in combination with GHCi's debugger.

### Further reading

<http://felsin9.de/nnis/ghc-vis>

### 6.4.4 Hat — the Haskell Tracer

Report by:	Olaf Chitil
------------	-------------

Hat is a source-level tracer for Haskell. Hat gives access to detailed, otherwise invisible information about a computation.

Hat helps locating errors in programs. Furthermore, it is useful for understanding how a (correct) program works, especially for teaching and program maintenance. Hat is not a time or space profiler. Hat can be used for programs that terminate normally, that terminate with an error message or that terminate when interrupted by the programmer.

Tracing a program with Hat consists of two phases: First the program needs to be run such that it additionally writes a trace to file. To add trace-writing, *hat-trans* translates all the source modules *Module* of a Haskell program into tracing versions *Hat.Module*. These are compiled as normal and when run the program does exactly the same as the original program except for additionally writing a trace to file. Second, after the program has terminated, you view the trace with a tool. Hat comes with several tools for selectively viewing fragments of the trace in different ways: *hat-observe* for Hood-like observations, *hat-trail* for exploring a computation backwards, *hat-explore* for freely stepping through a computation, *hat-detect* for algorithmic debugging, ...

Hat is distributed as a package on Hackage that contains all Hat tools and tracing versions of standard libraries. Currently Hat supports Haskell 98 plus some language extensions such as multi-parameter type classes and functional dependencies. For portability all viewing tools have a textual interface; however, many tools use some Unix-specific features and thus run on Unix / Linux / OS X, but not on Windows.

Hat was mostly built around 2000–2004 and then disappeared because of lack of maintenance. Now it is back and new developments have started.

The source-to-source transformation of *hat-trans* has been completely rewritten to use the *haskell-src-exts* parser. Thus small bugs of the old parser disappeared and in the future it will be easier to cover more Haskell language extensions. This work was released on Hackage as Hat 2.8.

When a traced program uses any libraries besides the standard Haskell 98 / 2010 ones, these libraries currently have to be transformed (in trusted mode). So the plan for the next release of Hat is to enable Hat to use trusted libraries without having to transform them.

Feedback on Hat is welcome.

### Further reading

- Initial website: <http://projects.haskell.org/hat>
- Hackage package: <http://hackage.haskell.org/package/hat>

### 6.4.5 Tasty

Report by:	Roman Cheplyaka
Status:	active development; in use

Tasty is a new testing framework for Haskell. It is heavily influenced by `test-framework`, but has a cleaner

code base and is more extensible.

It has support for the following “providers”:

- HUnit
- QuickCheck
- SmallCheck
- Hspec
- Golden tests (tests whose expected results are stored in files)

It is also easy to create your own, “domain specific”, test providers.

Another dimension of extensibility, which has been added very recently, is so-called “ingredients”. Ingredients represent different ways in which a test suite can be executed. Examples of tasks performed by the standard ingredients include:

- simply running the tests and reporting the results;
- listing the test names without running them;
- updating the reference values for golden tests.

Again, it’s very easy to write and plug in a custom ingredient. Perhaps you want to integrate your test suite with some CI system, or you prefer a different output format, or have some other crazy idea on your mind? Tasty can do that.

Tasty was written and released during the summer 2013, but is already being used by at least 8 open-source projects.

### Further reading

For more information about tasty and how to use it, please consult the README at <http://documentup.com/feuerbach/tasty>

## 7 Libraries, Applications, Projects

### 7.1 Language Features

#### 7.1.1 Conduit

Report by:	Michael Snoyman
Status:	stable

While lazy I/O has served the Haskell community well for many purposes in the past, it is not a panacea. The inherent non-determinism with regard to resource management can cause problems in such situations as file serving from a high traffic web server, where the bottleneck is the number of file descriptors available to a process.

Left fold enumerators have been the most common approach to dealing with streaming data without using lazy I/O. While it is certainly a workable solution, it requires a certain inversion of control to be applied to code. Additionally, many people have found the concept daunting. Most importantly for our purposes, certain kinds of operations, such as interleaving data sources and sinks, are prohibitively difficult under that model.

The conduit package was designed as an alternate approach to the same problem. The root of our simplification is removing one of the constraints in the enumerator approach. In order to guarantee proper resource finalization, the data source must always maintain the flow of execution in a program. This can lead to confusing code in many cases. In conduit, we separate out guaranteed resource finalization as its own component, namely the ResourceT transformer.

Once this transformation is in place, data producers, consumers, and transformers (known as Sources, Sinks, and Conduits, respectively) can each maintain control of their own execution, and pass off control via coroutines. The user need not deal directly with any of this low-level plumbing; a simple monadic interface (inspired greatly by the pipes package) is sufficient for almost all use cases.

Since its initial release, conduit has been through many design iterations, all the while keeping to its initial core principles. Since the last HCAR, we've released version 1.0. This release introduces a simplification of the public facing API, optimizing for the common use cases. This was a minor change, and the conduit ecosystem has already caught up. The package has been in a mature state for quite some time now, and can be relied upon for most streaming data needs.

There is a rich ecosystem of libraries available to be used with conduit, including cryptography, network communications, serialization, XML processing, and more. The Web Application Interface was the original motivator for creating the library, and continues

to use it for expressing request and response bodies between servers and applications. As such, conduit is also a major player in the Yesod ecosystem.

The library is available on Hackage. There is an interactive tutorial available on the FP Complete School of Haskell. You can find many conduit-based packages in the Conduit category on Hackage as well.

#### Further reading

- <http://hackage.haskell.org/package/conduit>
- <https://www.fpcomplete.com/user/snoyberg/library-documentation/conduit-overview>
- <http://hackage.haskell.org/packages/archive/pkg-list.html#cat:conduit>

#### 7.1.2 lens

Report by:	Edward Kmett
Participants:	many others
Status:	very actively developed

The **lens** package provides families of lenses, isomorphisms, folds, traversals, getters and setters. That is to say, it provides a rich, compositional vocabulary for separating “what you want to do” from “what you want to do it to” built upon rigorous foundations.

Compared to other libraries that provide lenses, key distinguishing features for lens are that it comes “batteries included” with many useful lenses for the types commonly used from the Haskell Platform, and with tools for automatically generating lenses and isomorphisms for user-supplied data types.

Also, included in this package is a variant of Neil Mitchell's **uniplate** generic programming library, modified to provide a Traversal and with its combinators modified to work with arbitrary traversals.

Moreover, you do not need to incur a dependency on the lens package in order to supply (or consume) lenses or most of the other lens-like constructions offered by this package.

#### Further reading

- Simon Peyton Jones:  
<http://skillsmatter.com/podcast/scala/lenses-compositional-data-access-and-manipulation>
- Edward Kmett:  
<http://www.youtube.com/watch?v=cefnmjtAoIY>
- Lens Development, Visualized:  
<http://www.youtube.com/watch?v=ADAprOOgi-A&feature=youtu.be&hd=1>
- <http://hackage.haskell.org/package/lens>
- <http://lens.github.io/>

- o <https://github.com/ekmett/lens/wiki>
- o <https://github.com/ekmett/lens/issues>
- o <http://statusfailed.com/blog/2013/01/26/haskells-strength-generalising-with-lenses.html>
- o <http://ocharles.org.uk/blog/posts/2012-12-09-24-days-of-hackage-lens.html>
- o <http://www.haskellforall.com/2013/05/program-imperatively-using-haskell.html>
- o <https://www.fpcomplete.com/school/to-infinity-and-beyond/pick-of-the-week/a-little-lens-starter-tutorial>
- o <http://stackoverflow.com/questions/5767129/lenses-fclabels-data-accessor-which-library-for-structure-5769285#5769285>
- o <http://comonad.com/reader/2012/mirrored-lenses/>
- o <http://r6.ca/blog/20121209T182914Z.html>
- o <http://r6.ca/blog/20120623T104901Z.html>

### 7.1.3 folds

Report by: Edward Kmett  
 Status: actively developed

This package provides a playground full of resumable comonadic folds and folding homomorphisms between them.

#### Further reading

- o <http://hackage.haskell.org/package/folds>
- o <https://www.fpcomplete.com/user/edwardk/cellular-automata/part-2>
- o <http://squiring.blogspot.com/2008/11/beautiful-folding.html>
- o <http://conal.net/blog/posts/another-lovely-example-of-type-class-morphisms>
- o <http://conal.net/blog/posts/more-beautiful-fold-zipping>
- o <http://www.haskellforall.com/2013/08/composable-streaming-folds.html>

### 7.1.4 machines

Report by: Edward Kmett  
 Participants: Anthony Cowley, Shachaf Ben-Kiki, Paul Chiusano, Nathan van Doorn  
 Status: actively developed

*Ceci n'est pas une pipe*

This package exists to explore the design space of streaming calculations. Machines are demand-driven input sources like pipes or conduits, but can support multiple inputs.

You design a Machine by writing a Plan. You then construct the machine from the plan.

Simple machines that take one input are called a Process. More generally you can attach a Process to the output of any type of Machine, yielding a new Machine. More complicated machines provide other ways

of connecting to them.

Typically the use of machines proceeds by using simple plans into machine Tees and Wyes, capping many of the inputs to those with possibly monadic sources, feeding the rest input (possibly repeatedly) and calling run or runT to get the answers out.

There is a lot of flexibility when building a machine in choosing between empowering the machine to run its own monadic effects or delegating that responsibility to a custom driver.

#### Further reading

- o <https://vimeo.com/77164337>
- o <http://acowley.github.io/NYHUG/FunctionalRobotist.pdf>
- o <https://github.com/runarorama/scala-machines>
- o <https://dl.dropbox.com/u/4588997/Machines.pdf>

### 7.1.5 exceptions

Report by: Edward Kmett  
 Participants: Gabriel Gonzales, Michael Snoyman, John Weigley, Mark Lentczner, Alp Mestanogullari, Fedor Gogolev, Merijn Verstraaten, Matvey B. Aksenov  
 Status: actively developed

This package was begun as an effort to define a standard way to deal with exception handling in monad transformer stacks that could scale to the needs of real applications in terms of handling asynchronous exceptions, could support GHC now that *block* and *unblock* have been removed from the compiler, and which we could reason about the resulting behavior, and still support *mocking* on monad transformer stacks that are not built atop IO.

#### Further reading

<http://hackage.haskell.org/package/exceptions>

### 7.1.6 tables

Report by: Edward Kmett  
 Participants: Nathan van Doorn, Tim Dixon, Niklas Haas, Dag Odenhall, Petr Pilar, Austin Seipp  
 Status: actively developed

The `tables` package provides a multiply-indexed in-memory data store in the spirit of `ixset` or `data-store`, but with a `lens`-based API.

#### Further reading

- o <http://hackage.haskell.org/package/tables>
- o <https://github.com/ekmett/tables#examples>

## 7.2 Education

### 7.2.1 Exercism: crowd-sourced code reviews on daily practice problems

Report by:	Bob Ippolito
Status:	available

Exercism.io is an open source (AGPL) site that provides programming exercises suitable for new programmers, or programmers new to a programming language.

The feature that differentiates exercism from self-study is that once a solution is submitted, others who have completed that exercise have an opportunity to provide code review. Anecdotally, this seems to put programmers on the right track quickly, especially with regard to the subtler points of Haskell style, non-strict evaluation, and GHC-specific features.

Exercism fully supports Haskell as of August 2013, with more than 50 exercises currently available. As of this writing, 165 people have completed at least one Haskell exercise.

I intend to continue actively participating in the code review process and ensure that the Haskell exercise path is well maintained.

#### Further reading

<http://exercism.io/>

### 7.2.2 Talentbuddy

Report by:	Andrei Soare
Status:	available

Talentbuddy is a fun way for developers to practice their skills.

It offers access to a diverse selection of problems and it makes it easy to get feedback from your peers once you solve them.

You can write code in multiple languages — including Haskell — and if you get stuck, you can ask for help. The members of the community that already solved the challenge get to see your question and suggest ways to make progress.

Once you solve a problem, you get access to everyone else's solutions. By comparing your solution with the other ones you acquire knowledge about alternative strategies for solving the problem, or more elegant ways to write the solution.

Another great way to learn on Talentbuddy is by asking your peers to review your solution. Their reviews help increase your solution's readability and elegance.

#### Further reading

<http://www.talentbuddy.co/>

### 7.2.3 Holmes, Plagiarism Detection for Haskell

Report by:	Jurriaan Hage
Participants:	Brian Vermeer, Gerben Verburg

Holmes is a tool for detecting plagiarism in Haskell programs. A prototype implementation was made by Brian Vermeer under supervision of Jurriaan Hage, in order to determine which heuristics work well. This implementation could deal only with Helium programs. We found that a token stream based comparison and Moss style fingerprinting work well enough, if you remove template code and dead code before the comparison. Since we compute the control flow graphs anyway, we decided to also keep some form of similarity checking of control-flow graphs (particularly, to be able to deal with certain refactorings).

In November 2010, Gerben Verburg started to reimplement Holmes keeping only the heuristics we figured were useful, basing that implementation on `haskell-src-extends`. A large scale empirical validation has been made, and the results are good. We have found quite a bit of plagiarism in a collection of about 2200 submissions, including a substantial number in which refactoring was used to mask the plagiarism. A paper has been written, which has been presented at CSERC'13, and should become available in the ACM Digital Library.

The tool will be made available through Hackage at some point, but before that happens it can already be obtained on request from Jurriaan Hage.

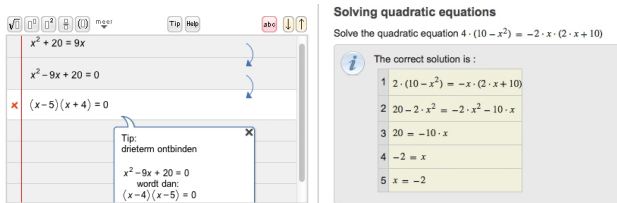
#### Contact

[J.Hage@uu.nl](mailto:J.Hage@uu.nl)

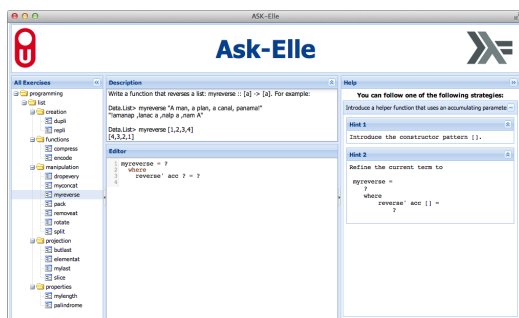
### 7.2.4 Interactive Domain Reasoners

Report by:	Bastiaan Heeren
Participants:	Johan Jeuring, Alex Gerdes, Josje Lodder
Status:	experimental, active development

The IDEAS project at Open Universiteit Nederland and Utrecht University aims at developing domain reasoners for stepwise exercises on various topics. These reasoners assist students in solving exercises incrementally by checking intermediate steps, providing feedback on how to continue, and detecting common mistakes. The reasoners are based on a strategy language, from which feedback is derived automatically. The calculation of feedback is offered as a set of web services, enabling external (mathematical) learning environments to use our work. We currently have a binding with the Digital Mathematics Environment of the Freudenthal Institute (first/left screenshot), the ActiveMath learning system of the DFKI and Saarland University (second/right screenshot), and our own online exercise assistant that supports rewriting logical expressions into disjunctive normal form.



We have continued working on the Ask-Elle functional programming tutor. This tool lets you practice introductory functional programming exercises in Haskell. The tutor can both guide a student towards developing a correct program, as well as analyze intermediate, incomplete programs and check whether or not certain properties are satisfied. We have recently extended the tutor with QuickCheck properties for testing the correctness of student programs, and for the generation of counterexamples. Normalization of functional programs is used for dealing with all kinds of variations in programs; this is ongoing research. We also want to make it as easy as possible for teachers to add programming exercises to the tutor, and to adapt the behavior of the tutor by disallowing or enforcing particular solutions. Teachers can adapt feedback by annotating the model solutions of an exercise. The tutor has an improved web-interface and is used in an introductory FP course at Utrecht University.



We have modeled the artificial intelligence of a real-time video game on top of the strategy combinator language used in the domain reasoners. In the future we expect to develop a serious game for communication skills using a similar approach.

The library for developing domain reasoners with feedback services is available as a Cabal source package. We have written a tutorial on how to make your own domain reasoner with this library. We have also released our domain reasoner for mathematics and logic as a separate package.

### Further reading

- Online exercise assistant (for logic), accessible from our [project page](#).
- Bastiaan Heeren, Johan Jeuring, and Alex Gerdes. [Specifying Rewrite Strategies for Interactive Exercises](#). Mathematics in Computer Science, 3(3):349–370, 2010.

- Johan Jeuring, Alex Gerdes, and Bastiaan Heeren. [A Programming Tutor for Haskell](#). Lecture Notes Central European School on Functional Programming, (CEFP 2011). Try our tutor at <http://ideas.cs.uu.nl/FPtutor/>.
- Tom Hastjarjanto, Johan Jeuring, and Sean Leather. [A DSL for describing the artificial intelligence in real-time video games](#). Third International Workshop on Games and Software Engineering (GAS 2013).

## 7.3 Parsing and Transforming

### 7.3.1 FliPpr

Report by:	Kazutaka Matsuda
Participants:	Meng Wang
Status:	experimental, prototype

FliPpr (“flip” + “ppr” (pretty-printer)) is a program transformation tool that takes a pretty-printing program and returns a parser consistent with the pretty-printer. It is common that, when we implement a programming language, we have to write a pair of programs: a pretty-printer and a parser, and it is expected that, especially early in the development, the syntax of the language changes frequently. In such a case, we have to update both the pretty-printer and the parser so that they continue to work with each other. FliPpr removes this maintenance burden from the programmers through program inversion techniques such that a consistent parser (in the sense that pretty-printed code is always correctly parsed) is automatically generated.

Pretty-printers in FliPpr are written with Wadler’s pretty-printing combinators as recursive functions on an AST datatype, which is very similar to what we normally do in Haskell. The differences are that there is a new combinator for embedding additional information for effective parsing, and there are syntactic restrictions in place. For details, please see our ESOP paper (doi: 10.1007/978-3-642-37036-6\_6).

Currently, the implementation is an experimental prototype. So bugs and unhelpful error messages are expected. However, you shall be able to play with the system by using the examples from the implementation page below.

### Further reading

<http://www-kb.is.s.u-tokyo.ac.jp/~kztk/FliPpr/>

### 7.3.2 epub-metadata

Report by:	Dino Morelli
Status:	experimental, actively developed

Library for parsing and manipulating epub OPF package data. Now with epub3 support.



- o Added support for epub3 documents. This was done using a single set of datatypes, not specific to either epub2 or epub3.
- o Redesigned the book file querying API to be an edsl. Actions are to be combined together based on what the developer needs from the document.
- o Data structures to contain epub metadata “sections” were redesigned to no longer be nested. Part of this change includes a typeclass-based pretty-print API for displaying this data.
- o Documentation rewrites and additions, including a working code example in the API docs.

epub-metadata is available from Hackage and the Darcs repository below.

See also `epub-tools` ( $\rightarrow$  7.9.2).

### Further reading

- o Project page:  
<http://ui3.info/d/proj/epub-metadata.html>
- o Source repository: `darcs get`  
<http://ui3.info/darcs/epub-metadata>

### 7.3.3 Utrecht Parser Combinator Library: uu-parsinglib

Report by:	Doaitse Swierstra
Status:	actively developed

With respect to the previous version the code for building interleaved parsers was split off into a separate package `uu-interleaved`, such that it can be used by other parsing libraries too. Based on this another small package `uu-options` was constructed which can be used to parse command line options and files with preferences. The internals of these are described in a technical report: <http://www.cs.uu.nl/research/techreps/UU-CS-2013-005.html>.

As an example of its use we show how to fill a record from the command line. We start out by defining the record which is to hold the options to be possibly set:

```
data Prefers = Agda | Haskell deriving Show
data Address = Address { city_ :: String
                      , street_ :: String }
                      deriving Show
data Name = Name { name_ :: String
                 , prefers_ :: Prefers
                 , ints_ :: [Int]
                 , address_ :: Address }
                 deriving Show
$(deriveLenses '' Name)
$(deriveLenses '' Address)
```

The next thing to do is to specify a default record containing the default values:

```
defaults = Name "Doaitse" Haskell []
          (Address "Utrecht"
                "Princetonplein")
```

Next we define the parser for the options, by specifying each option:

```
oName =
  name 'option' ( "name", pString,
                 "Name")
<> ints 'options' ( "ints", pNaturalRaw,
                   "Some numbers")
<> prefers 'choose' [( "agda", Agda,
                      "Agda preferred")
                    , ("haskell", Haskell,
                      "Haskell preferred")
                    ]
<> address 'field'
  ( city 'option' ("city", pString,
                 "Home city")
  <> street 'option' ("street", pString,
                    "Home Street")
  )
```

Finally when running this parser by the command `run (($defaults) <$> mkP oName)` on the string `("-int=7 -city=Tynaarlo -i 5 -agda -i3 " ++ "-street=Zandlust")` the result is

```
Name { name_ = Doaitse
      , prefers_ = Agda
      , ints_ = [7, 5, 3]
      , address_ = Address
                { city_ = Tynaarlo
                , street_ = Zandlust }
      }
```

If you make a mistake in the list of options, automatic error reporting and correction steps in and you get the following message:

```
./OptionsDemo --street=Zandlust -nDoaitse
-i3 --city=Tynaarlo
--name      [Char] optional Name
--ints      Int   recurring Some numbers
Choose at least one from(
--agda      required Agda preferred
--haskell    required Haskell preferred
)
--city      [Char] optional Home city
--street    [Char] optional Home Street
--
-- Correcting steps:
--   Inserted "-a" at position 70
--   expecting one of
--     [ "--agda", "--agda=", "--haskell",
--       "--haskell=", "--ints=", "--ints",
--       "-i", "-h", "-a" ]
--   Inserted EOT at position 70
--   expecting EOT
```

## Features

- Combinators for easily describing parsers which produce their results online, do not hang on to the input and provide excellent error messages. As such they are “surprise free” when used by people not fully aware of their internal workings.
- Parsers “correct” the input such that parsing can proceed when an erroneous input is encountered.
- The library basically provides the to be preferred applicative interface and a monadic interface where this is really needed (which is hardly ever).
- No need for *try*-like constructs which make writing `Parsec` based parsers tricky.
- Scanners can be switched dynamically, so several different languages can occur intertwined in a single input file.
- Parsers can be run in an interleaved way, thus generalizing the merging and permuting parsers into a single applicative interface. This makes it e.g. possible to deal with white space or comments in the input in a completely separate way, without having to think about this in the parser for the language at hand (provided of course that white space is not syntactically relevant).

## Future plans

Future versions will contain a check for grammars being not left-recursive, thus taking away the only remaining source of surprises when using parser combinator libraries. This makes the library even greater for use in teaching environments. Future versions of the library, using even more abstract interpretation, will make use of computed look-ahead information to speed up the parsing process further.

## Contact

If you are interested in using the current version of the library in order to provide feedback on the provided interface, contact [doaitse@swierstra.net](mailto:doaitse@swierstra.net). There is a low volume, moderated mailing list which was moved to [parsing@lists.science.uu.nl](mailto:parsing@lists.science.uu.nl) (see also <http://www.cs.uu.nl/wiki/bin/view/HUT/ParserCombinators>).

### 7.3.4 Grammar Products

Report by:	Christian Höner zu Siederdisen
Status:	usable, active development

We have developed a theory of algebraic operations over linear and context-free grammars. This theory allows us to combine simple “atomic” grammars to create more complex ones.

With the compiler that accompanies our theory, we make it easy to experiment with grammars and their

products. Atomic grammars are user-defined and the algebraic operations on the atomic grammars are embedded in a rigorous mathematical framework.

Our immediate applications are problems in computational biology and linguistics. In these domains, algorithms that combine structural features on individual inputs (or tapes) with an alignment or structure between tapes are becoming more commonplace. Our theory will simplify building grammar-based applications by dealing with the intrinsic complexity of these algorithms.

We provide multiple types of output.  $\text{\LaTeX}$  is available to those users who prefer to manually write the resulting grammars. Alternatively, Haskell modules can be created. `TemplateHaskell` and `QuasiQuoting` machinery is also available turning this framework into a fully usable embedded domain-specific language. The DSL or Haskell module use `ADPfusion` ( $\rightarrow$  7.11.1) with multitape extensions, delivering “close-to-C” performance.

### Further reading

- <http://www.bioinf.uni-leipzig.de/Software/gramprod/>
- [http://dx.doi.org/10.1007/978-3-319-02624-4\\_8](http://dx.doi.org/10.1007/978-3-319-02624-4_8)

### 7.3.5 HERMIT

Report by:	Andy Gill
Participants:	Andy Gill, Andrew Farmer, Ed Komp, Neil Sculthorpe, Adam Howell, Robert Blair, Ryan Scott, Patrick Flor, Michael Tabone
Status:	active

The Haskell Equational Reasoning Model-to-Implementation Tunnel (HERMIT) is an NSF-funded project being run at KU ( $\rightarrow$  9.9), which aims to improve the applicability of Haskell-hosted Semi-Formal Models to High Assurance Development. Specifically, HERMIT uses a Haskell-hosted DSL and a new refinement user interface to perform rewrites directly on Haskell Core, the GHC internal representation.

This project is a substantial case study of the application of `Worker/Wrapper` on larger examples. In particular, we want to demonstrate the equivalences between efficient Haskell programs, and their clear specification-style Haskell counterparts. In doing so there are several open problems, including refinement scripting and managing scaling issues, data representation and presentation challenges, and understanding the theoretical boundaries of the `worker/wrapper` transformation.

We have reworked KURE (<http://www.haskell.org/communities/11-2008/html/report.html#sect5.5.7>), a Haskell-hosted DSL for strategic programming, as the basis of our rewrite capabilities, and constructed the rewrite kernel making use of the GHC Plugins

architecture. A journal writeup of the KURE internals has been submitted to JFP, and is available on the group webpage. As for interfaces to the kernel, we currently have a command-line REPL, and an Android version is under development. Thus far, we have used HERMIT to successfully mechanize many smaller examples of program transformations, drawn from the literature on techniques such as concatenate vanishes, tupling transformation, and worker/wrapper. We are scaling up our capabilities, and working on larger examples.

HERMIT has also been used in two larger case studies. The first, led by Michael Adams from Portland State University in Oregon, uses HERMIT to mechanize the optimization of scrap your boilerplate generics, leading to execution speeds that were as fast as hand-optimized code (→ 7.4.1). The second uses HERMIT to implement a custom GHC optimization pass which enables fusion of nested streams within the Stream Fusion framework.

### Further reading

<http://www.ittc.ku.edu/csdl/fpg/Tools/HERMIT>

#### 7.3.6 haskell-names

Report by:	Roman Chepyaka
Status:	active development; in use

`haskell-src-exts` has long been used for Haskell analysis tools. Unfortunately, without proper semantic information about the names, these tools are either very imprecise, or have to perform the tricky name resolution process themselves.

`haskell-names` solves exactly this problem. It is a Haskell name resolution library built on top of `haskell-src-exts`.

In the simplest case, you give it an AST produced by `haskell-src-exts`'s parser, and it gives you back the same AST annotated with the name binding information.

Besides that, there's a more powerful interface to name resolution, using open recursion. It's essentially a generic traversal of the AST, where at each node the algebra has access to the global and lexical name environments. This is described in more detail in the article "Open your name resolution".

`haskell-names` is also integrated with `cabal`, thanks to the `haskell-packages` library (→ 7.3.7). You can install packages using the special `hs-gen-iface` compiler, and `haskell-names` will be aware of them.

Some of the missing features are detection of certain kinds of scoping errors, recording and applying fixities, and resolution of type variables. Some language extensions are not yet fully supported.

Still, `haskell-names` can correctly resolve most of the valid Haskell code and is being successfully used by the Fay and Ariadne (→ 6.1.3) projects.

### Further reading

For further directions, please see the README at <http://documentup.com/haskell-suite/haskell-names>. To learn more about "open name resolution", see <http://ro-che.info/articles/2013-03-04-open-name-resolution>

#### 7.3.7 haskell-packages

Report by:	Roman Chepyaka
Status:	active development; in use

If you are writing a Haskell compiler, you typically want to integrate it with Cabal (→ 6.3.1), to be able to build ordinary Haskell packages.

If you go the hard way, this involves:

1. Parsing command line parameters
2. Teaching Cabal how to call your compiler (which typically involves patching Cabal)
3. Package management

`haskell-packages` solves most of these problems for you. All you need to do is to provide the function to do actual compilation and tell a couple of other things about your compiler.

The Cabal patch which includes support for a compiler based on `haskell-packages` has been recently merged upstream and will be part of the next major Cabal release.

### Further reading

<http://documentup.com/haskell-suite/haskell-packages>

#### 7.3.8 parsers

Report by:	Edward Kmett
Participants:	Nathan Filardo, Dag Odenall, Mario Blazevic, Tony Morris, Tim Dixon, Greg Fitzgerald
Status:	actively developed

This package provides a common lingua franca for working with `parsec`-like parsing combinator libraries, such that the combinators support being lifted over monad transformers. Instances are provided for use with the `parsec Parser` and `base's ReadP`, and it is used by `trifecta` (→ 7.3.9) to provide its suite of parsing combinators.

Notably, many of the combinators have been modified to only require the use of `Alternative` rather than `MonadPlus`, enabling some base `Parser` instances to operate more efficiently.

### Further reading

<http://hackage.haskell.org/package/parsers>

### 7.3.9 trifacta

Report by:	Edward Kmett
Participants:	Austin Seipp, Nathan Filardo, John Weigley
Status:	actively developed

This package is designed to explore the space of “human scale” parsers for programming languages. That is to say, it isn’t optimized for parsing protocols or other huge streaming datasets, but rather to provide nice error messages for files that are usually written by hand by human beings.

Trifacta supports `clang`-style colored diagnostics with markup denoting locations, spans and fixits for user code. It builds on top of the `parsers` (→ 7.3.8) framework for most of its parsing combinators.

Much of the focus of `trifacta` is on supporting functionality beyond basic parsing, such as syntax highlighting, that arise once you have a programming language.

In the long term, we plan to support built-in CPP, auto-completion and parser transformers to support Haskell-style layout.

#### Further reading

<http://hackage.haskell.org/package/trifacta>

## 7.4 Generic and Type-Level Programming

### 7.4.1 Optimising Generic Functions

Report by:	José Pedro Magalhães
Participants:	Michael D. Adams, Andrew Farmer
Status:	actively developed

Datatype-generic programming increases program reliability by reducing code duplication and enhancing reusability and modularity. However, it is known that datatype-generic programs often run slower than type-specific variants, and this factor can prevent adoption of generic programming altogether. There can be multiple reasons for the performance penalty, but often it is caused by conversions to and from representation types that do not get eliminated during compilation.

Fortunately, it is also known that generic functions can be specialised to concrete datatypes, removing any overhead from the use of generic programming. We have investigated compilation techniques to specialise generic functions and remove the performance overhead of generic programs in Haskell. We used a representative generic programming library and inspected the generated code for a number of example generic functions. After understanding the necessary compiler optimisations for producing efficient generic code, we benchmarked the runtime of our generic functions against handwritten variants, and concluded that all the overhead can indeed be removed automatically by

the compiler. More details can be found in the IFL’12 paper linked below.

We have also investigated how to optimise the popular Scrap Your Boilerplate (SYB) generic programming library. Using a HERMIT (→ 7.3.5) script for implementing an optimisation pass in the compiler, we have removed all runtime overhead from SYB functions. More details can be found in the draft paper linked below.

#### Further reading

- [Optimisation of Generic Programs through Inlining](#)
- [Optimizing SYB Is Easy!](#)

### 7.4.2 traverse-with-class

Report by:	Roman Cheplyaka
Participants:	Sjoerd Visscher
Status:	experimental

`traverse-with-class` is an advanced generic programming library.

It is most closely related to `syb-with-class` (“Scrap your boilerplate with class”). The main difference is using `gtraverse` instead of `gfoldl`:

`class GTraversable (c :: Constraint) a where`

```
gtraverse
:: (Applicative f, ?c :: proxy c)
⇒ (∀d.c d ⇒ d → f d)
→ a → f a
```

This is based on the insight that `gtraverse` is in theory equivalent in power to `gfoldl`, but many instances can be encoded in a more direct way using `gtraverse`.

For example, the uniform instance for lists — one that treats all the list elements as being on the same level — is trivial to encode using `gtraverse`, and is very tricky to encode using `gfoldl`.

Another difference compared to `syb-with-class` is usage of modern Haskell extensions to simplify the API. We use constraint kinds instead of explicit dictionary types, and implicit parameters to simulate type lambda application.

`traverse-with-class` is the base of open recursive name resolution in `haskell-names` (→ 7.3.6).

#### Further reading

- `traverse-with-class` on hackage <http://hackage.haskell.org/package/traverse-with-class>
- [Generalizing generic fold](http://ro-che.info/articles/2013-03-11-generalizing-gfoldl.html) <http://ro-che.info/articles/2013-03-11-generalizing-gfoldl.html>
- [gtraverse vs gfoldl](http://ro-che.info/articles/2013-03-29-gtraverse-vs-gfoldl.html) <http://ro-che.info/articles/2013-03-29-gtraverse-vs-gfoldl.html>

### 7.4.3 constraints

Report by:	Edward Kmett
Participants:	Sjoerd Visscher, Austin Seipp
Status:	actively developed

This package provides data types and classes for manipulating values of kind `Constraint` as exposed by GHC since 7.4.

#### Further reading

- <http://hackage.haskell.org/package/constraints>
- <http://comonad.com/reader/2011/what-constraints-entail-part-1/>
- <http://comonad.com/reader/2011/what-constraints-entail-part-2/>

## 7.5 Mathematics

### 7.5.1 Rlang-QQ

Report by:	Adam Vogt
Status:	active development

Rlang-QQ is intended to make it easier to call R from Haskell programs. This allows access to a large number of R packages for graphing, statistics or other uses. Rlang-QQ provides a quasiquoter which runs the R interpreter and tries to translate values between the two languages.

Haskell expressions can be referenced from R using syntax like `$(take 10 [1.0 .. ])`. Haskell variables can also be passed in by prefixing them with `hs_`: `hs_x` refers to `x`. Values that can be taken out of a Haskell `x :: Chan t` are accessible using `ch_x`. When the R code has an assignment such as `hs_x <- f()`, the quasiquote evaluates to an `HList` record which contains the result from `f()`.

Future work may include supporting the serialization of more data types between the two languages, passing data between the two runtimes in-memory instead of through files, and doing inference when possible on the R-code to restrict the types of the Haskell values that are serialized or deserialized.

#### Further reading

- <http://hackage.haskell.org/package/Rlang-QQ>
- <http://www.r-project.org/>
- <http://www.haskell.org/haskellwiki/Quasiquotation>

### 7.5.2 order-statistics

Report by:	Edward Kmett
Status:	stable

This package extends Bryan O’Sullivan’s `statistics` package with support for order statistics and L-

estimators.

An order statistic is simply a position in the sorted list of samples given just the size of the sample. L-estimators are linear combinations of order-statistics.

L-estimators are used in robust statistics to collect statistics that are robust in the presence of outliers, and have the benefit that you can jackknife them without changing their asymptotics.

This package provides a compositional vocabulary for describing order statistics.

#### Further reading

- <http://hackage.haskell.org/package/order-statistics>
- [http://en.wikipedia.org/wiki/Order\\_statistic](http://en.wikipedia.org/wiki/Order_statistic)
- <http://en.wikipedia.org/wiki/L-estimator>

### 7.5.3 bed-and-breakfast

Report by:	Julian Fleischer
Status:	active development

Bed and breakfast is a pure Haskell linear algebra library that I built between getting out of the bed and having breakfast — thus the name. It features basic operations on matrices, for which it makes use of boxed and unboxed mutable arrays as necessary to improve performance — everything is pure though, as stateful computations happen in the `ST` monad. It currently excels in inverting matrices and finding determinants.

Bed and breakfast is published under the MIT license and available via hackage as `bed-and-breakfast`. You are more than welcome to suggest improvements. Development happens at <http://hub.darcs.net/scravy/bed-and-breakfast>.

### 7.5.4 Eliminating Redundancies in Linear Systems

Report by:	Philipp Kant
Status:	active

A recurring task in perturbative quantum field theory is the exact solution of very large systems of linear equations, where the coefficients are multivariate polynomials. The systems can contain hundreds of thousands of equations, where many of those equations are linearly dependent. In complicated cases, solving the system requires several months of CPU time.

ICE is a small Haskell program that detects which equations in a given set are linearly independent, so that the rest can be safely discarded before an attempt to solve the system. Thus, the time that would be spent processing redundant information can be saved.

The algorithm works by mapping the whole system homomorphically from the ring of multivariate polynomials to a finite field  $\mathbb{F}_p$ , where computations are cheap and the system can be solved fast using standard Gaussian elimination. By keeping track of the required row

permutations, the linearly independent equations are identified.

Future plans include to use multiple images in  $\mathbb{F}_p$  to solve the original system via rational function reconstruction. This would avoid the intermediate expression swell that is encountered when a linear system over multivariate polynomials is solved directly.

#### Further reading

<http://arxiv.org/abs/1309.7287>

### 7.5.5 linear

Report by:	Edward Kmett
Participants:	Anthony Cowley, Ben Gamari, Jake McArthur, John Weigley, Elliott Hird, Eric Mertens, Niklas Haas, Casey McCann
Status:	actively developed

This package provides ‘low-dimensional’ linear algebra primitives that are based explicitly on the notion that all vector spaces are free vector spaces, and so are isomorphic to functions from some basis to an underlying field. This lets us use representable functors, which are represented by such a basis to encode all of our linear algebra operations, and provides a natural encoding for dense vector spaces.

A nice *lens*-based API is provided that permits punning of basis vector names between different vector types.

#### Further reading

<http://hackage.haskell.org/package/linear>

### 7.5.6 algebra

Report by:	Edward Kmett
Status:	experimental

This package provides a large cross section of constructive abstract algebra.

Notable theoretical niceties include the fact that comonads form a *Monad*, linear maps form an *Arrow*, and this package bundles a rather novel notion of geometric coalgebra alongside the more traditional algebras and coalgebras.

#### Further reading

<http://hackage.haskell.org/package/algebra>

### 7.5.7 semigroups and semigroupoids

Report by:	Edward Kmett
Participants:	Nathan van Doorn, Mark Wright, Adam Curtis
Status:	stable

The `semigroups` package provides a standard location to obtain the notion of `Semigroup`.

The `semigroupoids` package provides the notion of a `Semigroupoid`, which is a `Category` that does not necessarily provide *id*. These arise in practice for many reasons in Haskell.

Notably, we cannot express a product category with the existing implementation of `Data Kinds`.

But more simply, there are many types for which their Kleisli category or Cokleisli category lacks identity arrows, because they lack *return* or *extract*, but could otherwise pass muster.

With `semigroupoids` 4.0, this package has now come to subsume the previous `groupoids` and `semigroupoid-extras` packages.

#### Further reading

- <http://hackage.haskell.org/package/semigroups>
- <http://hackage.haskell.org/package/semigroupoids>

### 7.5.8 Arithmetics packages (Edward Kmett)

Report by:	Edward Kmett
Participants:	Sjoerd Visscher, Austin Seipp, Daniel Bergey, Chris Schneider, Ben Gamari
Status:	actively developed

- The `compensated` package provides compensated arithmetic for when you need greater precision than the native floating point representation can provide. A `Compensated Double` has over 100 bits worth of effective significand. Unlike other “double double” variants in other languages, this construction can be iterated. A `Compensated (Compensated Double)` gives over 200 bits worth of precision.

However, not all `RealFloat` operations have yet been upgraded to work in full precision.

- The `approximate` package (with Sjoerd Visscher and Austin Seipp) provides a notion of approximate result values and intervals with log-domain lower bounds on confidence. It also provides fast piecewise-rational, but monotone increasing approximate versions of `log` and `exp` that execute many times faster than the native machine instructions that are suitable for use in machine learning.
- The `intervals` package (with Daniel Bergey and Chris Schneider) provides basic interval arithmetic. An `Interval` is a closed, convex set of floating point values.

We do not control the rounding mode of the end points of the interval when using floating point arithmetic, so be aware that in order to get precise containment of the result, you will need to use an underlying type with both lower and upper bounds like `CReal`.

- The `log-domain` package (with Ben Gamari) provides log domain floats, doubles and complex numbers with an emphasis on supporting probabilities biased towards conservative lower bounds.

#### Further reading

- <http://hackage.haskell.org/package/compensated>
- <http://hackage.haskell.org/package/approximate>
- <http://hackage.haskell.org/package/intervals>
- <http://hackage.haskell.org/package/log-domain>

#### 7.5.9 ad

Report by:	Edward Kmett
Participants:	Alex Lang, Takayuki Muranushi, Chad Scherrer, Lennart Augustsson, Ben Gamari, Christopher White
Status:	actively developed

This package provides an intuitive API for Automatic Differentiation (AD) in Haskell. Automatic differentiation provides a means to calculate the derivatives of a function while evaluating it. Unlike numerical methods based on running the program with multiple inputs or symbolic approaches, automatic differentiation typically only decreases performance by a small multiplier.

AD employs the fact that any program  $y = F(x)$  that computes one or more values does so by composing multiple primitive operations. If the (partial) derivatives of each of those operations is known, then they can be composed to derive the answer for the derivative of the entire program at a point.

This library contains at its core a single implementation that describes how to compute the partial derivatives of a wide array of primitive operations. It then exposes an API that enables a user to safely combine them using standard higher-order functions, just as you would with any other Haskell numerical type.

There are several ways to compose these individual Jacobian matrices. We hide the choice used by the API behind an explicit “Mode” type-class and universal quantification. This prevents the end user from exploiting the properties of an individual mode, and thereby potentially violating invariants or confusing infinitesimals.

We are actively seeking ways to better support unboxed vectors, new modes, new primitives, and better-optimized forms for gradient descent.

Features:

- Provides many variants on forward- and reverse-mode AD combinators with a common API.

- Type-level “branding” is used to both prevent the end user from confusing infinitesimals and to limit unsafe access to the implementation details of each mode.
- Each mode has a separate module full of combinators, with a consistent look and feel.

#### Further reading

- <http://hackage.haskell.org/package/ad>
- [http://en.wikipedia.org/wiki/Automatic\\_differentiation](http://en.wikipedia.org/wiki/Automatic_differentiation)
- <http://www.autodiff.org/>

#### 7.5.10 integration

Report by:	Edward Kmett
Participants:	Adrian Keet
Status:	actively developed

This package provides robust numeric integration via tanh-sinh quadrature. “Tanh-Sinh quadrature scheme is the fastest known high-precision quadrature scheme, especially when the time for computing abscissas and weights is considered. It has been successfully employed for quadrature calculations of up to 20,000-digit precision. It works well for functions with blow-up singularities or infinite derivatives at endpoints.”

#### Further reading

- <http://hackage.haskell.org/package/integration>
- [http://en.wikipedia.org/wiki/Tanh-sinh\\_quadrature](http://en.wikipedia.org/wiki/Tanh-sinh_quadrature)
- <http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/dhb-tanh-sinh.pdf>

#### 7.5.11 categories

Report by:	Edward Kmett
Participants:	Gwern Branwen
Status:	stable

This package provides a number of classes for working with Category instances with more structure in Haskell. In many ways this package can be viewed as an alternative to working with Arrows, as working with a CCC can provide you with much more fuel for optimization.

#### Further reading

<http://hackage.haskell.org/package/categories>

#### 7.5.12 contravariant

Report by:	Edward Kmett
Participants:	Dag Odenhall, Merijn Verstraaten
Status:	stable

This package provides the notion of a contravariant functor, along with various forms of composition for contravariant functors and Day convolution.

## Further reading

- <http://hackage.haskell.org/package/contravariant>
- <http://ncatlab.org/nlab/show/Day+convolution>

## 7.5.13 bifunctors

Report by:	Edward Kmett
Status:	stable

This package provides a standard location to retrieve the notion of a Bifunctor, Bifoldable or Bitraversable data type.

## Further reading

- <http://hackage.haskell.org/package/bifunctors>
- <http://ncatlab.org/nlab/show/bifunctor>

## 7.5.14 profunctors

Report by:	Edward Kmett
Participants:	Shachaf Ben-Kiki, Elliott Hird
Status:	stable

This package provides profunctors, which act like an Arrow you don't necessarily know how to put together.

These form the bedrock upon which *lens* ( $\rightarrow$  7.1.2) is built.

With profunctors 4.0 we've merged together the contents of the older `profunctors`, `profunctor-extras` and `representable-profunctors` packages.

In addition to the basic notion of a profunctor, we also provide the category of collages for a profunctor, notions of representable and corepresentable profunctors, along with weaker notions of Strong and Choice that correspond to various Arrow classes, profunctor composition.

## Further reading

- <http://hackage.haskell.org/package/profunctors>
- <http://blog.sigfpe.com/2011/07/profunctors-in-haskell.html>
- <https://www.fpcomplete.com/school/to-infinity-and-beyond/pick-of-the-week/profunctors>
- <http://ncatlab.org/nlab/show/profunctor>

## 7.5.15 comonad

Report by:	Edward Kmett
Participants:	Dave Menendez, Gabor Greif, David Luposchinsky, Sjoerd Visscher, Luke Palmer, Nathan van Doorn
Status:	stable

This package provides the comonads, the categorical dual of monads, along with comonad transformers, and the comonadic equivalent of the `mt1`.

With `comonad 4.0` we've merged together the contents of the older `comonad`, `comonad-transformers`,

and `comonads-fd` packages.

You can work with this package using Dominic Orchard's `codo-notation`, or use them directly.

The `kan-extensions` ( $\rightarrow$  7.5.17) package also provides a transformer that can turn a comonad into a monad.

## Further reading

- <http://hackage.haskell.org/package/comonad>
- [http://comonad.com/haskell/Comonads\\_1.pdf](http://comonad.com/haskell/Comonads_1.pdf)
- <http://www.cs.ox.ac.uk/ralf.hinze/WG2.8/28/slides/Comonad.pdf>
- <http://www.cl.cam.ac.uk/~dao29/publ/codo-notation-orchard-ifl12.pdf>
- <http://www.ioc.ee/~tarmo/papers/cmcs08.pdf>
- <http://cs.ioc.ee/~tarmo/papers/essence.pdf>

## 7.5.16 recursion-schemes

Report by:	Edward Kmett
Status:	stable

This package provides generalized bananas, lenses and barbed wire based on the recursion schemes that came out of the constructive algorithmics community over the years.

In addition to the standard recursion schemes, all of their distributive laws can be made compositional, enabling the creation of such interesting and impractical beasts as the zygo-histomorphic prepromorphism.

## Further reading

- <http://hackage.haskell.org/package/recursion-schemes>
- <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.174.8068&rep=rep1&type=pdf>
- <http://math.ut.ee/~eugene/kabanov-vene-mpc-06.pdf>
- <http://www.ioc.ee/~tarmo/tday-viinistu/kabanov-slides.pdf>
- <http://www.ioc.ee/~tarmo/papers/msfp08.pdf>
- <http://www.cs.uu.nl/wiki/pub/GP/Schedule/JoaoAlpuim.pdf>
- <http://eprints.eemcs.utwente.nl/7281/01/db-utwente-40501F46.pdf>
- <http://www.mii.lt/informatica/pdf/INFO141.pdf>
- <http://wwwhome.ewi.utwente.nl/~fokkinga/mmfphd.pdf>
- <http://comonad.com/reader/2008/elgot-coalgebras/>
- <http://comonad.com/reader/2008/time-for-chronomorphisms/>
- <http://comonad.com/reader/2008/dynamorphisms-as-chronomorphisms/>
- <http://comonad.com/reader/2008/generalized-hylomorphisms/>
- <http://web.engr.oregonstate.edu/~erwig/meta/>



- o <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.4.9706&rep=rep1&type=pdf>
- o <http://www.cs.ox.ac.uk/people/jeremy.gibbons/publications/metamorphisms-scp.pdf>

### 7.5.17 kan-extensions

Report by:	Edward Kmett
Status:	stable

This package provides Kan extensions, Kan lifts, various forms of the Yoneda lemma, and (co)density (co)monads.

These constructions have proven useful for many purposes:

- o Codensity can be used to accelerate the performance of code written for free monads or to correct the associativity of an “almost-monad” that fails the associativity law, as it performs a sort of fusion on ( $\gg$ ) operations.
- o CoT can be used to turn any Comonad into a Monad transformer.
- o Various forms of the Yoneda lemma give rise to ways to enforce “Functor fusion”.

#### Further reading

- o <http://hackage.haskell.org/package/kan-extensions>
- o <http://blog.sigfpe.com/2006/11/yoneda-lemma.html>
- o <http://blog.sigfpe.com/2006/12/yonedic-addendum.html>
- o <http://comonad.com/reader/2008/kan-extensions/>
- o <http://comonad.com/reader/2008/kan-extensions-ii/>
- o <http://comonad.com/reader/2008/kan-extension-iii/>
- o <http://blog.ezyang.com/2012/01/problem-set-the-codensity-transformation/>
- o <http://www.iai.uni-bonn.de/~jv/mpc08.pdf>
- o <http://www.cs.ox.ac.uk/ralf.hinze/Kan.pdf>
- o <http://ncatlab.org/nlab/show/Kan+lift>
- o <http://hackage.haskell.org/package/monad-ran>

## 7.6 Data Types and Data Structures

### 7.6.1 HList — A Library for Typed Heterogeneous Collections

Report by:	Oleg Kiselyov
Participants:	Adam Vogt, Ralf Lämmel, Kean Schupke

HList is a comprehensive, general purpose Haskell library for typed heterogeneous collections including extensible polymorphic records and variants. HList is analogous to the standard list library, providing a host of various construction, look-up, filtering, and iteration primitives. In contrast to the regular lists, elements of heterogeneous lists do not have to have the same type.

HList lets the user formulate statically checkable constraints: for example, no two elements of a collection may have the same type (so the elements can be unambiguously indexed by their type).

An immediate application of HLists is the implementation of open, extensible records with first-class, reusable, and compile-time only labels. The dual application is extensible polymorphic variants (open unions). HList contains several implementations of open records, including records as sequences of field values, where the type of each field is annotated with its phantom label. We and others have also used HList for type-safe database access in Haskell. HList-based Records form the basis of OOHaskell. The HList library relies on common extensions of Haskell 2010. HList is being used in AspectAG (<http://www.haskell.org/communities/11-2011/html/report.html#sect5.4.2>), typed EDSL of attribute grammars, and in HaskellDB.

The October 2012 version of HList library marks the significant re-write to take advantage of the fancier types offered by GHC 7.4 and 7.6. HList now relies on type-level booleans, natural numbers and lists, symbols, and on kind polymorphism. A number of operations are implemented as type functions. Another notable addition is unfold for heterogeneous lists. Many operations (projection, splitting) are now implemented in terms of unfold. Such a refactoring moved more computations to type-level, with no run-time overhead.

Since the last update, almost the entire HList has been rewritten in this new style. There are new operations: HReplicate, HScanr, HFoldl. Keyword function arguments become a part of the HList distribution. A new version of the Apply constraint, ApplyAB, improves the type inference obviating many type annotations. The Haddock documentation has been greatly extended, with many doctest.

#### Further reading

- o HList repository: <http://code.haskell.org/HList/>
- o HList: <http://okmij.org/ftp/Haskell/types.html#HList>
- o OOHaskell: <http://homepages.cwi.nl/~ralf/OOHaskell/>

### 7.6.2 Persistent

Report by:	Michael Snoyman
Participants:	Greg Weber, Felipe Lessa
Status:	stable

Persistent is a type-safe data store interface for Haskell. Haskell has many different database bindings available, but they provide few useful static guarantees. Persistent uses knowledge of the data schema to provide a type-safe interface that re-uses existing database binding libraries. Persistent is designed to work across different databases, and works on SQLite, PostgreSQL,

MongoDB, and MySQL, with an experimental backend for CouchDB.

The 1.2 release features a refactoring of the module hierarchy. We're taking this opportunity to clean up a few idiosyncracies in the API and make the documentation a bit more helpful, but otherwise the library is remaining unchanged.

The MongoDB backend features new helpers, query operators, and bug fixes for working with embedded/nested models. One can store a list of Maps or records inside a column/field. This is required for proper usage of MongoDB. In SQL an embedded object is stored as JSON, which is convenient as long as the column is not queried.

In order to accomodate various different backend types, Persistent is broken up into multiple components (separated by type classes). There is one for storage/serialization, one for uniqueness, and one for querying. This means that anyone wanting to create database abstractions can re-use the battle-tested persistent storage/serialization layer without having to implement the full query interface.

Persistent's query layer is the same for any backend that implement the query interface, although backends can define their own additional operators. The interface is a straightforward usage of combinators:

```
selectList [PersonFirstName == . "Simon",  
            PersonLastName == . "Jones"] []
```

There are some drawbacks to the query layer: it doesn't cover every use case. Persistent has built-in some very good support for raw SQL. One can run arbitrary SQL queries and get back Haskell records or types for single columns. In addition, Felipe Lessa has created a library called *esqueleto* for having complete control over generating SQL but with type safety. *persistent-MongoDB* also has helpers for working with raw queries.

### Future plans

Possible future directions for Persistent:

- Adding key-value databases such as Redis without a query layer.
- Full CouchDB support

Persistent users may also be interested in *Groundhog* (→7.6.3), a similar project.

Most of Persistent development occurs within the Yesod (→5.2.6) community. However, there is nothing specific to Yesod about it. You can have a type-safe, productive way to store data, even on a project that has nothing to do with web development.

### Further reading

- <http://www.yesodweb.com/book/persistent>
- <http://hackage.haskell.org/package/esqueleto>

## 7.6.3 Groundhog

Report by:	Boris Lykah
Status:	stable

*Groundhog* is a library for mapping user defined datatypes to the database and manipulating them in a high-level typesafe manner. It is easy to plug *Groundhog* into an existing project since it does not need modifying a datatype or providing detailed settings. The schema can be configured flexibly which makes integration with existing databases easy. It may have composite keys, indexes, references across several schemas. Typically, user can write only one line of settings to analyze the type and map it to the table. The migration mechanism can automatically check, initialize, and migrate database schema. *Groundhog* has backends for *Sqlite*, *PostgreSQL*, and *MySQL*.

Some parts of *Groundhog* API are similar to *Persistent* (→7.6.2), but there are major differences. The types can be polymorphic and contain multiple constructors. It allows creating sophisticated queries which might include arithmetic expressions, functions, and operators. Its support for the natural and composite keys is implemented using generic embedded datatype mechanism.

*Groundhog* has got at least two commercial users. Most of the recent changes were done to meet their needs. They include logging queries, references to tables not mapped to Haskell datatype, default column values, and several utility functions.

### Further reading

- Tutorial,  
<http://www.fpcomplete.com/user/lykahb/groundhog>
- Homepage, <http://github.com/lykahb/groundhog>
- Hackage package,  
<http://hackage.haskell.org/package/groundhog>

## 7.6.4 reflection

Report by:	Edward Kmett
Participants:	Elliott Hird, Oliver Charles, Carter Schonwald
Status:	stable

This package provides a mechanism to dynamically construct a type from a term that you can reflect back down to a term based on the ideas from “Functional Pearl: Implicit Configurations” by Oleg Kiselyov and Chung-Chieh Shan. However, the API has been implemented in a much more efficient manner.

This is useful when you need to make a typeclass instance that depends on a particular value in scope, such as a modulus or a graph.

### Further reading

- <http://hackage.haskell.org/package/reflection>

- o <http://www.cs.rutgers.edu/~ccshan/prepose/prepose.pdf>
- o <http://comonad.com/reader/2009/incremental-folds/>
- o <http://comonad.com/reader/2009/clearer-reflection/>
- o <https://www.fpcomplete.com/user/thoughtpolice/using-reflection>

### 7.6.5 tag-bits

Report by:	Edward Kmett
Status:	stable

This package provides access to the dynamic pointer tagging bits used by GHC, and can peek into infotables to determine (unsafely) whether or not a thunk has already been evaluated.

#### Further reading

- o <http://hackage.haskell.org/package/tag-bits>
- o <http://research.microsoft.com/en-us/um/people/simonpj/papers/ptr-tag/>
- o <http://ghc.haskell.org/trac/ghc/wiki/Commentary/Rts/HaskellExecution/PointerTagging>
- o <http://ghc.haskell.org/trac/ghc/wiki/Commentary/Rts/Storage/HeapObjects>

### 7.6.6 hyperloglog

Report by:	Edward Kmett
Participants:	Ozgun Ataman
Status:	actively developed

This package provides an approximate streaming (constant space) unique object counter.

Notably it can be used to approximate a set of several billion elements with 1-2% inaccuracy in around 1.5k of memory.

#### Further reading

- o <http://hackage.haskell.org/package/hyperloglog>
- o <http://algo.inria.fr/flajolet/Publications/FIFuGaMe07.pdf>

### 7.6.7 concurrent-supply

Report by:	Edward Kmett
Participants:	Andrew Cowie, Christiaan Baaij
Status:	stable

This package provides a fast supply of concurrent unique identifiers suitable for use within a single process. This benefits from greatly reduced locking overhead compared to `Data.Unique` as it only contents for the common pool every thousand or so identifiers.

One often has a desire to generate a bunch of integer identifiers within a single process that are unique within that process. You could use UUIDs, but they can be expensive to generate; you don't want to have

your threads contending for a single external counter if the identifier is not going to be used outside the process.

`concurrent-supply` builds a rose-tree-like structure which can be split; you can make smaller unique supplies and then you allocate from your supplies locally. Internally it pulls from a unique supply one block at a time as you walk into parts of the tree that haven't been explored. This ensures that computations are always replayable within a process, and that the result appears purely functional to an outside observer.

#### Further reading

<http://hackage.haskell.org/package/concurrent-supply>

### 7.6.8 hybrid-vectors

Report by:	Edward Kmett
Status:	actively developed

This package provides various ways in which you can mix the different types of `Vector` from Roman Leschinskiy's vector package to work with partially unboxed structures.

#### Further reading

- o <http://hackage.haskell.org/package/hybrid-vectors>
- o <https://www.fpcomplete.com/user/edwardk/revisiting-matrix-multiplication/part-3>

### 7.6.9 lca

Report by:	Edward Kmett
Participants:	Daniel Peebles, Andy Sonnenburg
Status:	actively developed

This package improves the previous known complexity bound of online lowest common ancestor search from  $O(h)$  to  $O(\log h)$  persistently, and without preprocessing by using skew-binary random-access lists to store the paths.

#### Further reading

- o <http://hackage.haskell.org/package/lca>
- o <https://www.fpcomplete.com/user/edwardk/online-lca>
- o <http://www.slideshare.net/ekmett/skewbinary-online-lowest-common-ancestor-search>

### 7.6.10 heaps

Report by:	Edward Kmett
Status:	actively developed

This package provides asymptotically optimal purely functional Brodal-Okasaki heaps with a "Haskelly" API.

## Further reading

- <http://hackage.haskell.org/package/heaps>
- <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.973>

## 7.6.11 sparse

Report by:	Edward Kmett
Participants:	Carter Schonwald
Status:	actively developed

This package provides sparse implicitly Morton-ordered matrices based on the series ‘revisiting matrix multiplication’ on the School of Haskell. It is efficient for sufficiently sparse matrices.

## Further reading

- <http://hackage.haskell.org/package/sparse>
- <https://www.fpcomplete.com/user/edwardk/revisiting-matrix-multiplication>

## 7.6.12 compressed

Report by:	Edward Kmett
Status:	stable

This package provides an LZ78-compressed stream as a data type in Haskell. Compression isn’t used directly for data compression, but rather to allow for the reuse of intermediate monoidal results when folding over the data set. LZ78 is rather distinctive among LZ-variants in that it doesn’t require exhaustively enumerating the token set or searching a window. By using conservative approximations of what possible values the stream may take, it is also possible to work with this LZ78 stream as an `Applicative` or `Monad` without sacrificing too much compression on the resulting unfolding.

A similar structure is provided for decompressing run-length encoded data efficiently by peasant exponentiation.

## Further reading

- <http://hackage.haskell.org/package/compressed>
- <http://oldwww.rasip.fer.hr/research/compress/algorithms/fund/lz/lz78.html>
- <http://www.binaryessence.com/dct/en000140.htm>

## 7.6.13 charset

Report by:	Edward Kmett
Status:	stable

This package provides fast unicode character sets based on complemented PATRICIA tries along with common charsets for variations on the posix standard and standard unicode blocks. This encoding has the benefit that a `CharSet` and its complement take the same amount of

space. This package is used as a building block by `parsers` (→ 7.3.8) and `trifecta` (→ 7.3.9).

## Further reading

<http://hackage.haskell.org/package/charset>

## 7.6.14 Convenience types (Edward Kmett)

Report by:	Edward Kmett
Participants:	several others
Status:	stable

- The `either` package provides an `EitherT` monad transformer, that unlike `ErrorT` does not carry the unnecessary class constraint. Removing this limitation is necessary for many operations.

`EitherT` is also used extensively by Gabriel Gonzales’ `errors` package.

With `either` 4.0, we consolidated many of the existing combinators from Chris Done’s `either` package and Gregory Crosswhite’s `either-unwrap` package, both of which are now deprecated.

- The `tagged` package provides a simple `Tagged` new-type that carries an extra phantom type parameter, and a `Proxy` data type that has since been merged into `base` with GHC 7.8.

These are useful as safer ways to plumb type arguments than by passing `undefined` values around.

- The `void` package provides a single “uninhabited” data type in a canonical location along with all of the appropriate instances.

The need for such a data type arises in shockingly many situations as it serves as an initial object for the category of Haskell data types.

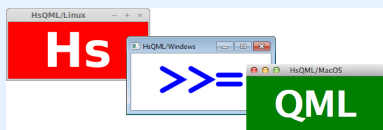
## Further reading

- <http://hackage.haskell.org/package/either>
- <http://hackage.haskell.org/package/errors>
- <http://hackage.haskell.org/package/tagged>
- <http://hackage.haskell.org/package/void>

## 7.7 User Interfaces

### 7.7.1 HsQML

Report by: Robin KAY  
Status: active development



HsQML provides access to a modern graphical user interface toolkit by way of a binding to the cross-platform Qt Quick framework.

The library focuses on mechanisms for marshalling data between Haskell and Qt’s domain-specific QML language. The intention is that QML, which incorporates both a declarative syntax and JavaScript code, can be used to design and animate the front-end of an application while being able to easily interface with Haskell code for functionality.

**Status** The latest version at time of press is 0.2.0.1. This supports Qt 4.7 or 4.8 only and has been tested on the major desktop platforms: Linux, Windows, and MacOS. The immediate plans for development are to migrate across to Qt 5.x as this is the future of the Qt platform.

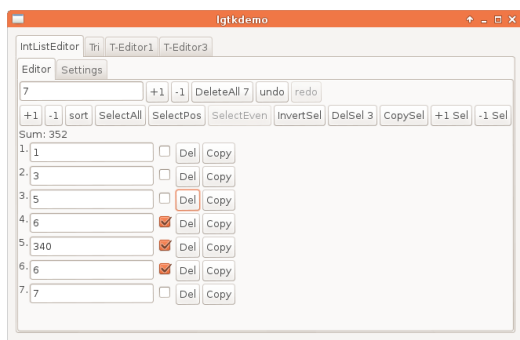
#### Further reading

<http://www.gekkou.co.uk/software/hsqml/>

### 7.7.2 LGtk: Lens-based Gtk API

Report by: Péter Diviánszky  
Status: first release, experimental, actively developed

Most Haskellers would like to use a mature FRP-based API for creating graphical user interfaces. But FRP may not be the best tool for special user interfaces, like interfaces which consist mainly of buttons, checkboxes, combo boxes, text entries, tabs and menus. The goal of the LGtk project is to give a lens-based API which better fits these user interfaces. LGtk is built on Gtk2Hs.



The first release of LGtk was announced on 15 April 2013. Currently the first version of the API is available with a demo application.

LGtk is being actively developed. I currently work on the following items:

- Give an approximation of the semantics of LGtk and adjust the API to the given semantics.
- Support for asynchronous effects.
- Make a tutorial for developers with lots of small examples.

#### Further reading

<http://people.inf.elte.hu/divip/LGtk/index.html>

### 7.7.3 Gtk2Hs

Report by: Daniel Wagner  
Participants: Axel Simon, Duncan Coutts, Andy Stewart, and many others  
Status: beta, actively developed

Gtk2Hs is a set of Haskell bindings to many of the libraries included in the Gtk+/Gnome platform. Gtk+ is an extensive and mature multi-platform toolkit for creating graphical user interfaces.

GUIs written using Gtk2Hs use themes to resemble the native look on Windows. Gtk is the toolkit used by Gnome, one of the two major GUI toolkits on Linux. On Mac OS programs written using Gtk2Hs are run by Apple’s X11 server but may also be linked against a native Aqua implementation of Gtk.

Gtk2Hs features:

- Automatic memory management (unlike some other C/C++ GUI libraries, Gtk+ provides proper support for garbage-collected languages)
- Unicode support
- High quality vector graphics using Cairo
- Extensive reference documentation
- An implementation of the “Haskell School of Expression” graphics API
- Bindings to many other libraries that build on Gtk: gio, GConf, GtkSourceView 2.0, glade, gstreamer, vte, webkit

Since the last release, there have been many bugfixes, and Peter Davies and Hamish Mackenzie have begun adding experimental Gtk3 support, enabled by the “gtk3” cabal flag.

#### Further reading

- News and downloads: <http://haskell.org/gtk2hs/>
- Development version: `darcs get`  
<http://code.haskell.org/gtk2hs/>

## 7.7.4 Haskell-EFL binding

Report by:	Sylvain Henry
Status:	Experimental

The Enlightenment Foundation Libraries (EFL) [1] provide a canvas with many rendering backends (X11, Windows, OpenGL, framebuffer, ...) as well as a unified interface to low-level OS dependent functionalities: user input (keyboard, mouse, multi-point, unicode texts with several input methods, ...), formatted text rendering, threads, timers, etc. Haskell-EFL is a binding to these C libraries whose aim is to provide a solid foundation to develop native Haskell widget toolkits.

In its current state, the binding is close to be complete for the modules we are interested in (namely Ecore and Evas). However it still needs some polishing and testing.

Source code for the binding is currently available on [2] and will be made available on Hackage as soon as it is considered complete and stable enough. Some examples using it can be found on [3].

In the short term, we plan to release a stable version on Hackage. In the medium term, we would like to develop a native replacement for the Edje library, that is an EDSL to create UI elements with themes, animations, data-binding (FRP), etc. Finally, the long term and more demanding goal is to develop a comprehensive set of reusable UI components.

### Further reading

- [1] <http://www.enlightenment.org/p.php?p=docs&l=en>
- [2] <https://github.com/hsyl20/graphics-efl>
- [3] <https://github.com/hsyl20/graphics-efl-examples>

## 7.7.5 threepenny-gui

Report by:	Heinrich Apfelmus
Status:	active development

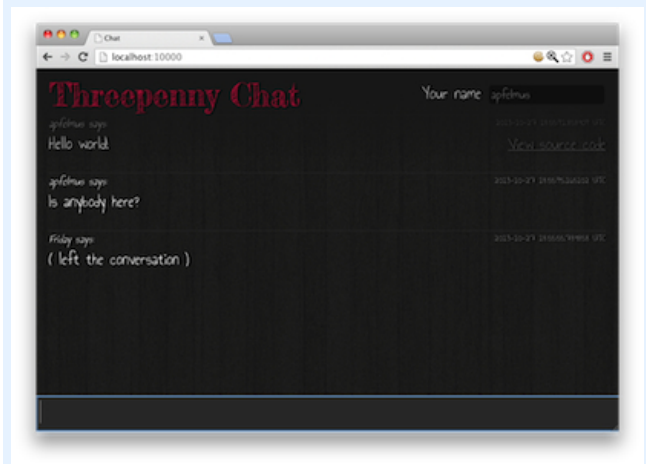
Threepenny-gui is a framework for writing graphical user interfaces (GUI) that uses the web browser as a display. Features include:

- *Easy installation.* Everyone has a reasonably modern web browser installed. Just install the library from Hackage and you are ready to go. The library is cross-platform.
- *HTML.* You have all capabilities of HTML at your disposal when creating user interfaces. This is a blessing, but it can also be a curse, so the library includes a few layout combinators to quickly create user interfaces without the need to deal with the mess that is CSS. A small JavaScript FFI allows you to include JS client libraries.

- *Functional Reactive Programming (FRP)* promises to eliminate the spaghetti code that you usually get when using the traditional imperative style for programming user interactions. Threepenny has an FRP library built-in, but its use is completely optional. Employ FRP when it is convenient and fall back to the traditional style when you hit an impasse.

### Status

The project is alive and kicking, version 0.3.0.1 is the latest release. You can download the library from Hackage and use it right away to write that cheap GUI you need for your project. Here a screenshot from the example code:



For a collection of real world applications that use the library, have a look at the gallery on the homepage. Daniel Austin's **FNIS**tash program is also featured in this report (→ 7.13.4).

### Current development

The library is still very much in flux, significant API changes are likely in future versions. The goal is to make GUI programming as simple as possible, and that just needs some experimentation.

The next version of **threepenny-gui** will include automatic garbage collection for HTML elements and it will (re-)introduce a UI monad that simplifies the JavaScript FFI and supports recursive uses of FRP.

### Further reading

- Project homepage: <http://haskell.org/haskellwiki/Threepenny-gui>
- Example code: <https://github.com/HeinrichApfelmus/threepenny-gui#examples>
- Application gallery: <http://haskell.org/haskellwiki/Threepenny-gui#Gallery>

## 7.7.6 reactive-banana

Report by: Heinrich Apfelmus  
Status: active development



Reactive-banana is a practical library for functional reactive programming (FRP).

FRP offers an elegant and concise way to express interactive programs such as graphical user interfaces, animations, computer music or robot controllers. It promises to avoid the spaghetti code that is all too common in traditional approaches to GUI programming.

The goal of the library is to provide a solid foundation.

- Writing *graphical user interfaces* with FRP is made easy. The library can be hooked into any existing event-based framework like wxHaskell or Gtk2Hs. A plethora of example code helps with getting started. You can mix FRP and imperative style. If you don't know how to express functionality in terms of FRP, just temporarily switch back to the imperative style.
- Programmers interested in implementing FRP will have a *reference* for a *simple semantics* with a working implementation. The library stays close to the semantics pioneered by Conal Elliott.
- It features an *efficient implementation*. No more spooky time leaks, predicting space & time usage should be straightforward.

*Status.* The latest version of the reactive-banana library is 0.7.1.3. Compared to the previous report, there has been no new public release as the API and its semantics have reached a stable plateau.

It turned out that the library suffered from a large class of space leaks concerning accumulated behaviors. This has been fixed in the development version, but not yet incorporated into a new release.

*Current development.* As foreshadowed in the last report, not much development has occurred on the reactive-banana library itself. Most of my efforts have been spent on the `threepenny-gui` project, which is a library for writing graphical user interfaces in Haskell (→7.7.5).

Fortunately, these development efforts are directly relevant to reactive-banana; graphical user interfaces have always been my main motivation for FRP in the first place. In particular, I have implemented a

new FRP module specifically for the `threepenny-gui` project, and this reimplemention has taught me many valuable lessons, which I hope to reintegrate into reactive-banana soon. The most important lesson is that the current API for reactive-banana is too complex — the type parameter `t` that indicates starting times just isn't worth it. I have also learned that recursion is best implemented differently from how I did it before, which leads to a fix for certain situations where reactive-banana couldn't handle recursion.

### Further reading

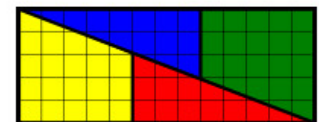
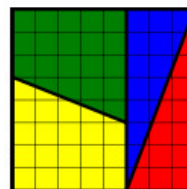
- Project homepage: <http://haskell.org/haskellwiki/Reactive-banana>
- Example code: <http://haskell.org/haskellwiki/Reactive-banana/Examples>
- `threepenny-gui`: <http://haskell.org/haskellwiki/Threepenny-gui>

## 7.8 Graphics and Audio

### 7.8.1 diagrams

Report by: Brent Yorgey  
Participants: Daniel Bergey, Jan Bracker, Daniil Frumin, Andy Gill, John Lato, Chris Mears, Jeff Rosenbluth, Michael Sloan, Ryan Yates  
Status: active development

The diagrams framework provides an embedded domain-specific language for declarative drawing. The overall vision is for diagrams to become a viable alternative to DSLs like MetaPost or Asymptote, but with the advantages of being *declarative*—describing what to draw, not how to draw it—and *embedded*—putting the entire power of Haskell (and Hackage) at the service of diagram creation. There is still much more to be done, but diagrams is already quite fully-featured, with a comprehensive user manual, a large collection of primitive shapes and attributes, many different modes of composition, paths, cubic splines, images, text, arbitrary monoidal annotations, named subdiagrams, and more.



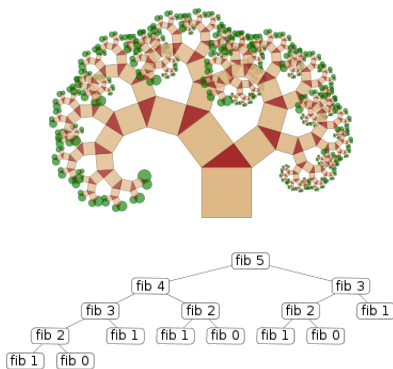
### What's new

Since the last HCAR edition, version 0.7 was released in August 2013. New features in 0.7 include:

- A big refactoring of the way segments, trails, and paths are represented; the new API is more powerful and semantically consistent.
- Functions to compute the curvature of path segments at a given point.
- Segment offsets (paths lying a constant distance away from a given segment). A fuller implementation with offsets for entire trails will be in the upcoming 1.0 release.
- A generalized color API, allowing backends to use whatever color space they want.
- Additions to the `diagrams-contrib` library, including a symmetric layout algorithm for binary trees, circle packing layout, a generalized turtle drawing interface, factorization diagrams, and iterated subset fractals.
- Many documentation improvements, using `diagrams-haddock` to generate example images.
- Big improvements to `diagrams-builder`, including much smarter rebuilding and `hsenv` support.
- Official support for the `SVGFonts` package, providing Haskell-native code for handling font data and converting strings into diagrams paths.

There have been many improvements and changes to the core diagrams libraries as well. A 1.0 release is planned for on or around November 20, to coincide with Brent's presentation at the New York Haskell Users' Group. Features slated for the 1.0 release include:

- A nice API for drawing arrows between arbitrary points or diagrams.
- Convenient integration with the `lens` ( $\rightarrow$  7.1.2) package.
- Path offsets and expansions: for example, the operation of "stroking" a path can now be internalized within diagrams, returning a closed path representing the outline of the stroke.
- Performance improvements: across-the-board improvements of around 30%, and more optimized SVG output.



## Contributing

There is plenty of exciting work to be done; new contributors are welcome! Diagrams has developed an encouraging, responsive, and fun developer community, and makes for a great opportunity to learn and hack on some "real-world" Haskell code. Because of its size, generality, and enthusiastic embrace of advanced type system features, diagrams can be intimidating to would-be users and contributors; however, we are actively working on new documentation and resources to help combat this. For more information on ways to contribute and how to get started, see the Contributing page on the diagrams wiki: <http://haskell.org/haskellwiki/Diagrams/Contributing>, or come hang out in the `#diagrams` IRC channel on freenode.



## Further reading

- <http://projects.haskell.org/diagrams>
- <http://projects.haskell.org/diagrams/gallery.html>
- <http://haskell.org/haskellwiki/Diagrams>
- <http://github.com/diagrams>
- <https://byorgey.wordpress.com/2012/08/28/creating-documents-with-embedded-diagrams/>
- <http://www.cis.upenn.edu/~byorgey/pub/monoid-pearl.pdf>
- <http://www.youtube.com/watch?v=X-8NcK2vOw>

## 7.8.2 csound-expression

Report by:	Anton Kholomiov
Status:	active, experimental

The `csound-expression` is a library for making electronic music with text. It can render a high level description of the music to Csound files. The Csound is used as an assembler for computer music.

The key aspect of the library is simplicity. A line of code should be enough to define the instrument, connect it to the midi-device and send the output to speakers. There are sensible defaults that allow the user of the library to express the musical ideas with very short sentences. As much as possible is derived from the context. There is a functional model that hides a low level wiring of the instruments. The FRP is used in the interface of the event streams.

There is a library of the cool instruments implemented in terms of the `csound-expression` primi-



tives. It's called the csound-catalog (<https://github.com/anton-k/csound-catalog>).

The library is available on Hackage (<http://hackage.haskell.org/package/csound-expression>) and github (<https://github.com/anton-k/csound-expression>).

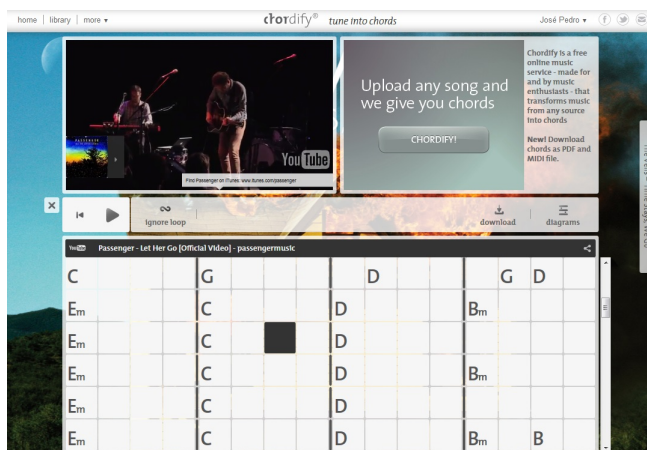
The future plans include the implementation of the Csound's GUI and testing of the library in the music applications.

#### Further reading

<https://github.com/anton-k/csound-expression>

### 7.8.3 Chordify

Report by:	José Pedro Magalhães
Participants:	W. Bas de Haas, Dion ten Heggeler, Gijs Bekenkamp, Tijmen Ruizendaal
Status:	actively developed



Chordify is a music player that extracts chords from musical sources like Soundcloud, Youtube, or your own files, and shows you which chord to play when. The aim of Chordify is to make state-of-the-art music technology accessible to a broader audience. Our interface is designed to be simple: everyone who can hold a musical instrument should be able to use it.

Behind the scenes, we use the sonic annotator for extraction of audio features. These features consist of the downbeat positions and the tonal content of a piece of music. Next, the Haskell program HarmTrace takes these features and computes the chords. HarmTrace uses a model of Western tonal harmony to aid in the chord selection. At beat positions where the audio matches a particular chord well, this chord is used in final transcription. However, in case there is uncertainty about the sounding chords at a specific position in the song, the HarmTrace harmony model will select the correct chords based on the rules of tonal harmony.

Chordify is free for everyone to use. We have recently added the ability to download PDF and MIDI

transcriptions of the chords for a small fee, and plan to release multiple new features soon. The code for HarmTrace is available on Hackage, and we have ICFP'11 and ISMIR'12 publications describing some of the technology behind Chordify.

#### Further reading

<http://chordify.net>

### 7.8.4 Euterpea

Report by:	Paul Hudak
Participants:	Donya Quick, Daniel Winograd-Cort
Status:	prototype release, active development

#### Overview

*Euterpea* is a Haskell library for computer music applications. It is a descendent of Haskore and HasSound, and is intended for both educational purposes as well as serious computer music development. *Euterpea* can be thought of as a “wide-spectrum” DSL, suitable for high-level music representation, algorithmic composition, and analysis; mid-level concepts such as MIDI; and low-level audio processing, sound synthesis, and instrument design. It also includes a *musical user interface* (MUI), a set of GUI widgets such as sliders, buttons, and so on.

The audio and MIDI-stream processing aspects of *Euterpea* are based on *arrows*, which makes programs analogous to signal processing diagrams. Using arrows prevents certain kinds of space leaks, and facilitates significant optimization strategies (in particular, the use of *causal commutative arrows*).

*Euterpea* is being developed at Yale in Paul Hudak's research group, where it has become a key component of Yale's new Computing and the Arts major. Hudak is teaching a two-term sequence in computer music using *Euterpea*, and is developing considerable pedagogical material, including a new textbook tentatively titled *The Haskell School of Music — From Signals to Symphonies* (HSoM). The name “*Euterpea*” is derived from “*Euterpe*”, who was one of the nine Greek Muses (goddesses of the arts), specifically the Muse of Music.

#### Status

The system is stable enough for experimental computer music applications, and for use in coursework either to teach Haskell programming or to teach computer music concepts.

All source code, papers, and a draft of the HSoM textbook can be found on the Yale Haskell Group website at: <http://haskell.cs.yale.edu/>.

#### History

*Haskore* is a Haskell library developed over 15 years ago by Paul Hudak and his students at Yale for high-level

computer music applications. *HasSound* was a later development that served as a functional front-end to csound's sound synthesis capabilities. Euterpea combines Haskell with a native Haskell realization of HasSound (i.e. no csound dependencies).

## Future Plans

Euterpea is a work in progress, as is the HSoM textbook. Computer-music specific MUI widgets (such as keyboards and guitar frets), further optimization strategies, better support for real-time MIDI and audio processing, and a parallel (multicore) implementation are amongst the planned future goals.

Anyone who would like to contribute to the project, please contact Paul Hudak at [paul.hudak@yale.edu](mailto:paul.hudak@yale.edu).

## FurtherReading

Please visit <http://haskell.cs.yale.edu/>. Click on “Euterpea” to learn more about the library, “Publications” to find our papers on computer music (including HSoM), and “CS431” or “CS432” to see the course material used in two computer music classes at Yale that use Euterpea.

## 7.9 Text and Markup Languages

### 7.9.1 Haskell XML Toolbox

Report by:	Uwe Schmidt
Status:	eighth major release (current release: 9.3)

#### Description

The Haskell XML Toolbox (HXT) is a collection of tools for processing XML with Haskell. It is itself purely written in Haskell 98. The core component of the Haskell XML Toolbox is a validating XML-Parser that supports almost fully the Extensible Markup Language (XML) 1.0 (Second Edition). There is a validator based on DTDs and a new more powerful one for Relax NG schemas.

The Haskell XML Toolbox is based on the ideas of HaXml and HXML, but introduces a more general approach for processing XML with Haskell. The processing model is based on arrows. The arrow interface is more flexible than the filter approach taken in the earlier HXT versions and in HaXml. It is also safer; type checking of combinators becomes possible with the arrow approach.

HXT is partitioned into a collection of smaller packages: The core package is `hxt`. It contains a validating XML parser, an HTML parser, filters for manipulating XML/HTML and so called XML pickler for converting XML to and from native Haskell data.

Basic functionality for character handling and decoding is separated into the packages `hxt-charproperties` and `hxt-unicode`. These

packages may be generally useful even for non XML projects.

HTTP access can be done with the help of the packages `hxt-http` for native Haskell HTTP access and `hxt-curl` via a libcurl binding. An alternative lazy non validating parser for XML and HTML can be found in `hxt-tagsoup`.

The XPath interpreter is in package `hxt-xpath`, the XSLT part in `hxt-xslt` and the Relax NG validator in `hxt-relaxng`. For checking the XML Schema Datatype definitions, also used with Relax NG, there is a separate and generally useful regex package `hxt-regex-xschema`.

The old HXT approach working with filter `hxt-filter` is still available, but currently only with `hxt-8`. It has not (yet) been updated to the `hxt-9` mayor version.

## Features

- Validating XML parser
- Very liberal HTML parser
- Lightweight lazy parser for XML/HTML based on Tagsoup (<http://www.haskell.org/communities/05-2010/html/report.html#sect5.11.3>)
- Binding to the expat parser via hexpat package
- Easy de-/serialization between native Haskell data and XML by pickler and pickler combinators
- XPath support
- Full Unicode support
- Support for XML namespaces
- Cabal package support for GHC
- HTTP access via Haskell bindings to libcurl and via Haskell HTTP package
- Tested with W3C XML validation suite
- Example programs
- Relax NG schema validator
- XML Schema validator (next release)
- Lightweight regex library with full support of Unicode and XML Schema Datatype regular expression syntax
- An HXT Cookbook for using the toolbox and the arrow interface
- Basic XSLT support
- GitHub repository with current development versions of all packages <http://github.com/UweSchmidt/hxt>

## Current Work

The master thesis and project implementing an XML Schema validator started in October 2011 has been finished. The validator will be released in a separate module `hxt-xschema`. Integration with `hxt` has been prepared in `hxt-9.3`. The XML Schema datatype library has also been completed, all datatypes including date and time types are implemented. But there is still a need for testing the validator, especially with the W3C

test suite. Hopefully testing will be done in the next few months. With the release of the schema validator the the master thesis will also be published on the HXT homepage. The current state of the validator can be found in the HXT repository on github.

### Further reading

The Haskell XML Toolbox Web page (<http://www.fh-wedel.de/~si/HXmlToolbox/index.html>) includes links to downloads, documentation, and further information.

The latest development version of HXT can be found on github under (<https://github.com/UweSchmidt/hxt>).

A getting started tutorial about HXT is available in the Haskell Wiki (<http://www.haskell.org/haskellwiki/HXT>). The conversion between XML and native Haskell data types is described in another Wiki page ([http://www.haskell.org/haskellwiki/HXT/Conversion\\_of\\_Haskell\\_data\\_from/to\\_XML](http://www.haskell.org/haskellwiki/HXT/Conversion_of_Haskell_data_from/to_XML)).

### 7.9.2 epub-tools (Command-line epub Utilities)

Report by:	Dino Morelli
Status:	stable, actively developed

A suite of command-line utilities for creating and manipulating epub book files. Included are: epubmeta, epubname, epubzip.

epubmeta is a command-line utility for examining and editing epub book metadata. With it you can export, import and edit the raw OPF Package XML document for a given book. Or simply dump the metadata to stdout for viewing in a friendly format.

epubname is a command-line utility for renaming epub ebook files based on the metadata. It tries to use author names and title info to construct a sensible name.

epubzip is a handy utility for zipping up the files that comprise an epub into an .epub zip file. Using the same technology as epubname, it can try to make a meaningful filename for the book.

This project is built on the latest epub-metadata library and so supports epub3 for the first time.

See also epub-metadata (→ 7.3.2).

epub-tools is available from Hackage and the Darcs repository below.

### Further reading

- Project page: <http://ui3.info/d/proj/epub-tools.html>
- Source repository: darcs get <http://ui3.info/darcs/epub-tools>

### 7.9.3 lens-aeson

Report by:	Edward Kmett
Participants:	Paul Wilson, Benno Fajstajck, Michael Sloan, Adrian Keet
Status:	actively developed

This package provides a suite of combinators that wrap around Bryan O’Sullivan’s `aeson` library using the lens library (→ 7.1.2) to make many data access and manipulation problems much more succinctly expressible. We provide lenses, traversals, isomorphisms and prisms that conspire to make it easy to manipulate complex JSON objects.

### Further reading

- <http://hackage.haskell.org/package/lens-aeson>
- <https://www.fpcomplete.com/user/tel/lens-aeson-traversals-prisms>

### 7.9.4 hyphenation

Report by:	Edward Kmett
Status:	stable

This package provides configurable Knuth-Liang hyphenation using the UTF-8 encoded hyphenation patterns for 69 languages, based on the patterns provided by the `hyph-utf8` project for  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ . It can be mixed with a pretty-printer to provide proper break-points within words.

### Further reading

- <http://hackage.haskell.org/package/hyphenation>
- <http://www.ctan.org/tex-archive/language/hyph-utf8>

## 7.10 Natural Language Processing

### 7.10.1 NLP

Report by:	Eric Kow
------------	----------

The Haskell Natural Language Processing community aims to make Haskell a more useful and more popular language for NLP. The community provides a mailing list, Wiki and hosting for source code repositories via the Haskell community server.

The Haskell NLP community was founded in March 2009. The list is still growing slowly as people grow increasingly interested in both natural language processing, and in Haskell.

### New packages

- *multext-east-msd 0.1.0.4* (Jan Snajder)

This package is an implementation of the MULTEXT-East morphosyntactic descriptors. It could be useful for work with Eastern European languages.

- *concraft 0.8* (Jakub Waszczuk)

Concraft is a morphological disambiguation library designed for highly-inflectional languages. It is based on conditional random fields extended with additional, position-wise restrictions on the output domain, which are used to impose consistency between the modeled label sequences and morphosyntactic analysis results (Waszczuk 2012; see further reading).

See also the package *concraft-pl*, a morphosyntactic tagging tool for the Polish language which relies on the Concraft library.

- *nerf 0.5.0* (Jakub Waszczuk)

The package provides a named entity (NE) recognition tool which can be used to model tree-like structures of NEs. It combines the IOB encoding method (used to translate between the original, forest representation of NEs and the sequence of atomic labels) with the sequence labeler based on linear-chain conditional random fields.

- *dawg 0.11* (Jakub Waszczuk)

The library implements directed acyclic word graphs internally represented as minimal acyclic deterministic finite-state automata. It provides fast insert and delete operations which can be used to build the automaton on-the-fly and a static hashing functionality. The library can be particularly useful to store language dictionaries (e.g. morphological dictionaries or resources of named entities). The implementation is not very efficient at the moment, but it provides a convenient map-like interface and should be easy to use.

### Updated packages

- *GenI 0.24.1* (Eric Kow)

GenI is a surface realiser (part of a natural language generation system) using Feature Based Tree Adjoining Grammar. This latest version can be customised to work with alternative semantic inputs, making it easier to integrate with wider applications.

- *sequor 0.4.2* (Grzegorz Chrupala)

Sequor is a sequence labeler based on Collins's (2002) perceptron. Sequor has a flexible feature template language and is meant mainly for NLP applications such as Named Entity labeling, Part of Speech tagging or syntactic chunking. This release includes the SemiNER named entity recognizer, with pre-trained models for German and English.

<https://bitbucket.org/gchrupala/sequor>

- *hiera 0.1.0.0* (Grzegorz Chrupala)

Hiera implements the algorithm for hierarchical clustering of word-class probability distributions described in Chrupala 2012 (see Further Reading): it is an agglomerative clustering algorithm where the distance between clusters is defined as the Jensen-Shannon divergence between the probability distributions over classes associated with each word-type.

<https://bitbucket.org/gchrupala/hiera>

At the present, the mailing list is mainly used to make announcements to the Haskell NLP community. We hope that we will continue to expand the list and expand our ways of making it useful to people potentially using Haskell in the NLP world.

### Further reading

- The Haskell NLP page <http://projects.haskell.org/nlp>
- Grzegorz Chrupala. 2012. *Hierarchical clustering of word class distributions*. NAACL-HLT 2012 Workshop on the Induction of Linguistic Structure.
- Jakub Waszczuk. 2012. *Harnessing the CRF complexity with domain-specific constraints. The case of morphosyntactic tagging of a highly inflected language*. In Proceedings of the 24th International Conference on Computational Linguistics (COLING 2012).

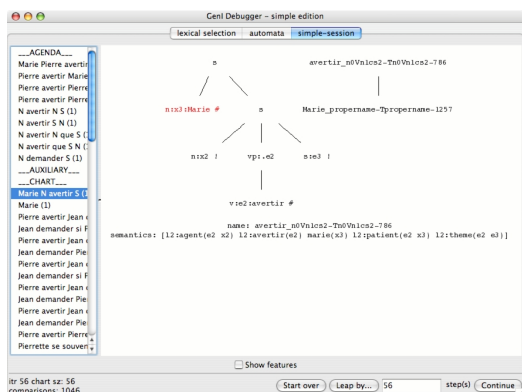
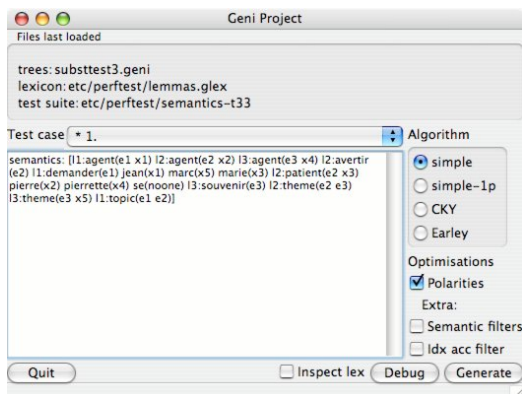
### 7.10.2 GenI

Report by:

Eric Kow

GenI is a surface realizer for Tree Adjoining Grammars. Surface realization can be seen a subtask of natural language generation (producing natural language utterances, e.g., English texts, out of abstract inputs). GenI in particular takes a Feature Based Lexicalized Tree Adjoining Grammar and an input semantics (a conjunction of first order terms), and produces the set of sentences associated with the input semantics by the grammar. It features a surface realization library, several optimizations, batch generation mode, and a graphical debugger written in wxHaskell. It was developed within the TALARIS project and is free software licensed under the GNU GPL, with dual-licensing available for commercial purposes.

GenI is now mirrored on GitHub, with its issue tracker and wiki and homepage also hosted there. The most recent release, GenI 0.24 (2013-09-18), allows for custom semantic inputs, making it simpler to use GenI in a wider variety for applications. This has recently been joined by a companion *geni-util* package which offers a rudimentary *geniserver* client and a reporting tool for grammar debugging.



GenI is available on Hackage, and can be installed via `cabal-install`, along with its GUI and HTTP server user interfaces. For more information, please contact us on the `geni-users` mailing list.

### Further reading

- <http://github.com/kowey/GenI>
- <http://projects.haskell.org/GenI>
- Paper from Haskell Workshop 2006: <http://hal.inria.fr/inria-00088787/en>
- <http://websympa.loria.fr/wwsympa/info/geni-users>

## 7.11 Bioinformatics

### 7.11.1 ADPfusion

Report by: Christian Höner zu Siederdisen  
 Status: usable, active development

ADPfusion provides a domain-specific language (DSL) for the formulation of dynamic programs with a special emphasis on computational biology. Following ideas established in Algebraic dynamic programming (ADP) a problem is separated into a grammar defining the search space and one or more algebras that score and select elements of the search space. The DSL has been designed with performance and a high level of abstraction in mind.

As an example, consider a grammar that recognizes palindromes. Given the non-terminal  $p$ , as well as parsers for single characters  $c$  and the empty input  $\epsilon$ ,

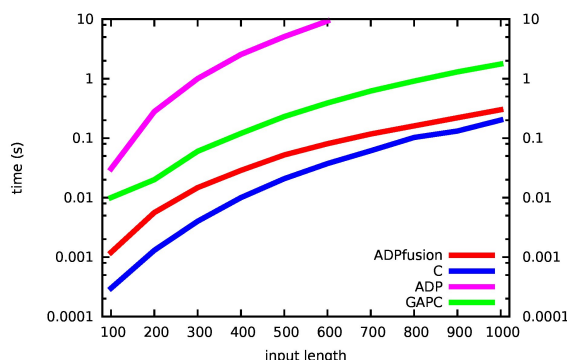
the production rule for palindromes can be formulated as  $p \rightarrow c p c \mid \epsilon$ .

The corresponding ADPfusion code is similar:

```
(p, f <<< c % p % c ||| g <<< e ... h)
```

We need a number of combinators as “glue” and additional evaluation functions  $f$ ,  $g$ , and  $h$ . With  $f c_1 p c_2 = p \ \&\& \ (c_1 \equiv c_2)$  scoring a candidate,  $g e = \text{True}$ , and  $h xs = \text{or } xs$  determining if the current substring is palindromic.

As of now, code written in ADPfusion achieves performance close to hand-optimized C, and outperforms similar approaches (Haskell-based ADP, GAPC producing C++) thanks to stream fusion. The figure shows running times for the *Nussinov algorithm*.



Starting with ADPfusion 0.2, dynamic programs on *more than one input* sequence can be written. This allows efficient dynamic programs that compute, say, the alignment of two or more inputs. More complicated algorithms of coupled context-free grammars also become possible with this new, *multi-dimensional* expansion. Together with generalised index spaces, more algorithms can be implemented efficiently, while at the same time reducing the effort required to implement these more complicated algorithms *correctly*.

### Further reading

- <http://www.tbi.univie.ac.at/~choener/adpfusion>
- <http://hackage.haskell.org/package/ADPfusion>
- <http://dx.doi.org/10.1145/2364527.2364559>

### 7.11.2 *Ab-initio* electronic structure in Haskell

Report by: Alessio Valentini  
 Participants: Felipe Zapata, Angel Alvarez  
 Status: Active

We are three friends from Alcalá de Henares (Spain), two PhD students in computational chemistry from ResMol group and one sysadmin working at Alcalá University computer center. We all share the same passion in programming and after some adventures in Fortran, Bash, Python and Erlang we are now fully committed to Haskell. As PhD students working in this area, every

day we face codes that are both difficult to read and improve, with no guidelines and poor documentation.

The set of problems inherent in computational chemistry are mainly due to the theoretical models complexity and the need of reducing as much as possible the computational time, leading to a demand of extremely solid and efficient software. What is happening in the actual context is the result of a poor interface between the two adjoining worlds of chemist and computer science and the necessity of publishing papers and scientific material to raise funds. This usually leads to software hastily developed by a few chemists with only a side-interest in programming and therefore a limited skill set.

The very few software that can be deemed remarkable are usually the result of massive funding, and even those packages are now facing huge problems in terms of parallelization, concurrency and stability of the code. Most of the efforts are spent trying to fix these issues instead of being addressed at developing better code (improve modularity and intelligibility) or new features and new algorithms.

We witness the proliferation of projects that serve no other purpose than to provide a bridge between different software, while the main core of molecular modeling codes, in most cases written in Fortran 77, remains untouched since the eighties.

Our first purpose in this project is to become better at Haskell programming and having fun managing a package that is every day becoming bigger. But we kind of dream of a molecular modeling software that can fully express the great upsides of functional programming. Fewer lines of code, better readability, great parallelization, embedded domain specific languages (EDSL) ... and maybe more efficiency, too !

*Ab-initio* molecular modeling is a branch of computational chemistry that, for a set of given atoms, solves the Schrödinger equation (the analogous of Newton's equation in quantum mechanics), with no inclusion of parameters derived from experimental data. In such systems it is quite easy to calculate forces between nuclei but things get tricky when we calculate the potential energy contribution of forces related to electrons. In this case we can adopt a first approximation, the so called Hartree-Fock, that considers the electron-electron repulsion as an *average* between each electron and the mean field of all the others. This theory is right now the cornerstone of more sophisticated methods, such *Multiconfigurational Methods*, *Møller-Plesset Perturbation Theory* or *Coupled Cluster*, and the mathematical models behind its implementation are vastly used throughout the world of computational chemistry.

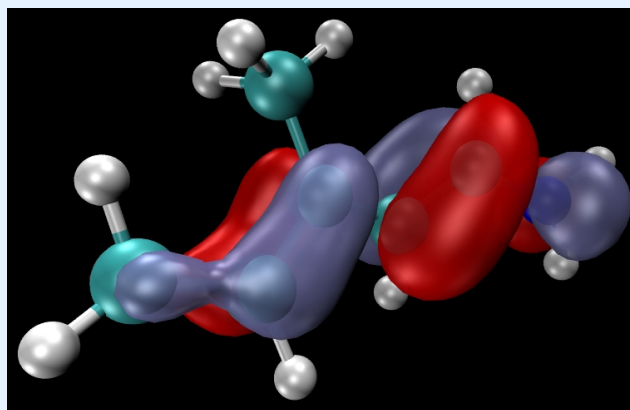
This package can calculate the Hartree Fock energy of a given molecule geometry and a basis set solving the *Roothaan Hall* equations through a self consistent field procedure. It uses the *Harris Functional* as an initial

density guess and the *DIIS* method to greatly improve the convergence.

The entire code is written using the *Repa* library and focusing our efforts on efficiency, parallelism (speedups vs cores: 2,3 on 4 and 3.5 on 8) and code readability. Using Haskell's higher order abstraction we are trying to develop an EDSL appropriate for quantum mechanics problems, creating code operators able to fairly mimic the physical ones.

The code is available for download in [Felipe's gitHub page](#).

*A Hartree Fock  $\pi$  orbital in PSB3:*



We are currently developing this code in our spare time, working on analytical gradients, on the *Prisma algorithm* and on a *solid eigenvalue problem solver*. The aims of this projects are a full Haskell implementation of *Multiconfigurational Methods* and possibly an integration with our *molecular dynamics project*.

#### Further reading

- <https://github.com/felipeZ/Haskell-abinitio.git>
- <http://themonadreader.files.wordpress.com/2013/03/issue214.pdf>

### 7.11.3 Semi-Classical Molecular Dynamics in Haskell

Report by:	Alessio Valentini
Participants:	Felipe Zapata, Angel Alvarez
Status:	Active

As a first approximation, we can split the world of Molecular Dynamics into three branches: Force Fields, Classical (Semi-Classical) and Quantum Molecular Dynamics. The first approach completely ignores the description of the electrons, and the system is described by a "Balls and Springs" model leading to very cheap calculations that can be performed in big systems.

From a chemical point of view, anyway, this approach often suffers severe drawbacks, since every time an accurate description of electrons is needed (i.e. when studying the formation or breaking of bonds, reactions involving excited states, or heavily polarized systems)

we cannot rely on pure Classical Mechanics.

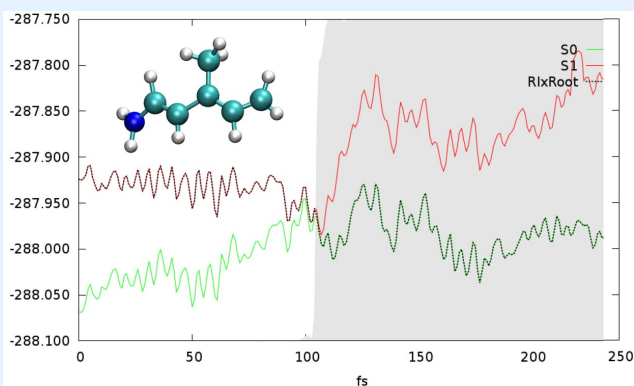
On the other side, even if the Quantum Dynamics approach is capable of describing the real quantum behavior of every single electron and nucleus, it comes with a huge increase in computational cost. It is basically unaffordable for systems with more than 5-6 atoms. That's why we need to take in consideration the Classical and Semi Classical Dynamics, where the system's forces are calculated using a Quantum method, while the geometry is updated with Classical Mechanics and some *ad-hoc formulae* to take into account quantum effects.

As PhD students in computational chemistry we often found ourselves in this situation: we have a chemical problem that might appear simple at first, but then it is usually quite difficult to find all the features necessary to tackle it in the same package. It is often the case where software "X" is lacking feature "Z" while software "Y" is missing feature "W".

The possible solutions to this impasse are:

1. to encode the missing features in the software of choice, a task that can reveal itself as very difficult and time consuming, since most of the time we are dealing with monolithic software written in Fortran, usually quite old and poorly maintained.
2. to write an external software (i.e. parser/launcher) capable of interact concurrently with several software, which is currently the approach employed in most cases. So much that the vast majority of computational chemists keeps a personal folder that contains just collections of parsers and scripts.

*Energies vs time for a two electronic states system:*



Our project takes advantage of the exceptional modularity that Haskell offers, and represents our effort to unify in a comprehensive tool all those routines that are needed in our research group to perform Classical and Semi Classical Molecular Dynamics. Our current goal is to keep a robust code and to minimize the need to use external software, limiting their application to the computation of the gradient.

Given some initial conditions and an external program (currently Molcas and Gaussian are supported)

capable of calculating the energy gradient, our code is able to parse its log file and perform the whole "Semi-Classical part" of the Molecular Dynamics.

The code employs the Velocity Verlet algorithm to propagate the geometries, the Nosé Hoover thermostat for a constant temperature bath and the Tully Hammes Schiffer hopping algorithm (along with correction of Persico-Granucci) to take in consideration hops between different electronic states. It also features the possibility to add external forces to the molecule, to simulate constrained conditions that can be found, for example, in a protein binding pocket.

This is still a small project, but we are using it constantly in our research group as a flexible tool for molecular dynamics, waiting for our other project to calculate the ab-initio gradient for us.

#### Further reading

<https://github.com/AngelitoJ/HsDynamics>

#### 7.11.4 Biohaskell

Report by:	Ketil Malde, Michal J. Gajda
Participants:	Christian Höner zu Siederdisen, Nick Ignolia, Felipe Almeida Lessa, Dan Fornika, Maik Riechert, Ashish Agarwal, Grant Rotskoff



Bioinformatics in Haskell is a steadily growing field, and the *Bio* section on Hackage now contains 53 libraries and applications. The biohaskell web site coordinates this effort, and provides documentation and related information. Anybody interested in the combination of Haskell and bioinformatics is encouraged to sign up to the mailing list (currently by emailing Ketil), and to register and document their contributions on the <http://biohaskell.org> wiki.

BioHaskell now contains a fledgling collection of libraries for structural analysis of biomolecules. hPDB is a parallel parser of Protein DataBank file format, which parses the largest structures faster than single-threaded Python code, and faster than parallel BioJava parser. It uses an octree data structure for fast querying of atom positions and contacts, and Iterable class to allow for iteration over deeply contained objects within hierarchical collections. The author of the library, Michal

J. Gajda plans to soon release other libraries for processing biomolecular data, in particular `parseSTAR` for parsing STAR\* format used for BMRB database of nuclear magnetic resonance data, and `hDat` library for processing small-angle scattering data.

### Further reading

- <http://biohaskell.org>
- <http://blog.malde.org>
- <http://hackage.haskell.org/package/hPDB>
- <http://hackage.haskell.org/package/Octree>
- <http://hackage.haskell.org/package/lterable>
- <http://www.tbi.univie.ac.at/~choener/haskell/>
- <http://adp-multi.ruhoh.com>

### 7.11.5 arte-ephys: Real-time electrophysiology

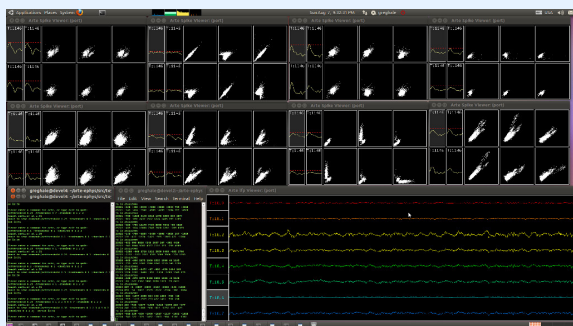
Report by:	Greg Hale
Participants:	Alex Chen, Sarah Walker
Status:	work in progress

Arte-ephys is a soft real-time neural recording system for experimental systems neuroscientists.

Our lab uses electrode arrays for brain recording in freely moving animals, to determine how these neurons build, remember, and use spatial maps.

We previously recorded and analyzed our data in two separate stages. We are now building a recording system focused on streaming instead of offline analysis, for real-time feedback experiments. For example, we found that activity in the brain of resting rats often wanders back to representations of specific parts of a recently-learned maze, and we would now like to automatically detect these events and reward the rat immediately for expressing them, to see if this influences either the speed of learning of a specific part of the maze or the nature of later spatial information coding.

We have a working (though outdated) backend system written in C++ that communicates with analog signal acquisition boards, performs simple signal conditioning, and broadcasts nearly-raw data through ZMQ. Our work on real-time neuron extraction, position-tracking, and neural information decoding in Haskell is still preliminary. Our immediate goal is to implement the high-level decoding system (which we can test using already-recorded data), and then to implement the lower-level data acquisition components piece-by-piece.



We enthusiastically welcome participation from anyone interested in seeing Haskell creep into the domain of wet-lab science.

### Further reading

- <http://github.com/ImAlsoGreg/arte-ephys>
- <http://github.com/ImAlsoGreg/haskell-tetrode-ephys>
- <http://web.mit.edu/wilsonlab/html/research.html>

## 7.12 Embedding DSLs for Low-Level Processing

### 7.12.1 Feldspar

Report by:	Emil Axelsson
Status:	active development

Feldspar is a domain-specific language for digital signal processing (DSP). The language is embedded in Haskell and is currently developed at Chalmers University of Technology (→ 9.8).

The motivating application of Feldspar is telecoms processing, but the language is intended to be useful for DSP in general. The aim is to allow DSP functions to be written in pure functional style in order to raise the abstraction level of the code and to enable more high-level optimizations. The current version consists of an extensive library of numeric and array processing operations as well as a code generator producing C code for running on embedded targets.

At present, Feldspar can express the data-intensive numeric algorithms which are at the core of any DSP application. There is also support for the expression and compilation of parallel algorithms. As future work remains to extend the language to deal with interaction with the environment (e.g., processing of streaming data) and to support compilation to heterogeneous multi-core targets.

### Further reading

- <https://github.com/Feldspar/feldspar-language>
- <http://hackage.haskell.org/package/feldspar-language>
- <http://hackage.haskell.org/package/feldspar-compiler>

### 7.12.2 Kansas Lava

Report by:	Andy Gill
Participants:	Andy Gill, Bowe Neuenschwander
Status:	ongoing

Kansas Lava is a Domain Specific Language (DSL) for expressing hardware descriptions of computations, and is hosted inside the language Haskell. Kansas Lava programs are descriptions of specific hardware entities, the connections between them, and other computational



abstractions that can compile down to these entities. Large circuits have been successfully expressed using Kansas Lava, and Haskell's powerful abstraction mechanisms, as well as generic generative techniques, can be applied to good effect to provide descriptions of highly efficient circuits.

- The Fabric monad is now a Monad transformer. The Fabric monad historically provided access to named input/output ports, and now also provides named variables, implemented by ports that loop back on themselves. This additional primitive capability allows for a *typed* state machine monad. This design gives an elegant stratospheric pattern: purely functional circuits using streams; a monad for layout over *space*; and a monad for state generation, that acts over *time*.
- On top of the Fabric monad, we are implementing an atomic transaction layer, which provides a BSV-like interface, but in Haskell. An initial implementation has been completed, and this is being reworked to include BSV's Ephemeral History Registers.

### Further reading

<http://www.ittc.ku.edu/csdl/fpg/Tools/KansasLava>

## 7.13 Others

### 7.13.1 General framework for multi-agent systems

Report by:	Nickolay Kudasov
Status:	experimental

The goal is to create a general framework for developing and testing of multi-agent systems. That includes general representation for multi-agent systems as well as library implementations for well-known agent models, distributed algorithms and communication and coordination patterns.

Notions of agent and environment are separated with the help of free monads. Agent-environment interface is defined by an underlying functor.

The basic representation of agent and environment has been chosen and tested for an agent-based distributed graph coloring problem.

The concrete implementation is being revised frequently and thus is not very stable.

Implementations for some general distributed algorithms (ABT, DBA, etc.) will be available shortly.

### Further reading

<https://github.com/fizruk/free-agent>

### 7.13.2 ersatz

Report by:	Edward Kmett
Participants:	Johan Kiviniemi, Iain Lane
Status:	stable

Ersatz is a library for generating QSAT (CNF/QBF) problems using a monad. It takes care of generating the normal form, encoding your problem, marshaling the data to an external solver, and parsing and interpreting the result into Haskell types.

What differentiates Ersatz from other SAT bindings is the use of observable sharing in the API.

This enables you to use the a much richer subset of Haskell than the purely monadic meta-language, and it becomes much easier to see that the resulting encoding is correct.

Support is offered for decoding various Haskell datatypes from the solution provided by the SAT solver.

A couple of examples are included with the distribution. Neither are as fast as a dedicated solver for their respective domains, but they showcase how you can solve real world problems involving 10s or 100s of thousands of variables and constraints.

### Further reading

<http://hackage.haskell.org/package/ersatz>

### 7.13.3 Clckwrks

Report by:	Jeremy Shaw
------------	-------------

clckwrks (pronounced "clockworks") is a blogging and content management system (CMS). It is intended to compete directly with popular PHP-based systems. Pages and posts are written in markdown and can be edited directly in the browser. The system can be extended via plugins and themes packages.

At present, clckwrks is still alpha, and requires Haskell knowledge to install and configure. However, the goal is to create an end user system that requires zero Haskell knowledge. It will be possible to one-click install plugins and themes and perform all other administrative functions via the browser.

### Future plans

We are currently focused on four tasks:

1. Overhaul of the plugin system to support one-click installation of plugins and themes
2. Improvements to the user experience in the core blogging and page editing functionality
3. Simplifying installation
4. Improved documentation

Once the core is solid, we will focus development efforts on creating plugins to extend the core functionality.

### Further reading

<http://www.clckwrks.com/>

#### 7.13.4 FNISstash

Report by:	Daniel Austin
------------	---------------

FNISstash is a utility application for the PC game Torchlight 2 (TL2). It presents a graphical interface to allow the user to manipulate the game's inventory stash files, store/retrieve items in a database external to the game, search the item database by key phrase(s), and generates reports identifying which items have yet to be found. The project began at the end of 2012 and first release was in Sept 2013. Several subsequent releases have been made to fix bugs and implement requests from users.

FNISstash is configured by modifying a file to point to TL2's game asset pack and save files. At first time start up, many of the game's graphical and interface assets are decompressed from the asset pack and written to disk using the Devil bindings. The save files are de-scrambled and parsed using logic that was backwards-engineered by the author, and each item is registered in an SQLite3 database file for efficient searching and storage. The GUI presented to the user utilizes the `threepenny-gui` package (→ 7.7.5). A web backend runs in a console window and users interact with the GUI through their web browser. The decision to use `threepenny-gui` was motivated by the need to easily set up, build, and deploy a GUI on the Windows platform. A video demo of the first release is available on the homepage.

FNISstash was originally conceived as an exercise to learn Haskell by building a program that is actually useful instead of fruitlessly reading journal papers and reddit posts. The experience was highly challenging, educational, and enjoyable.

### Further reading

[http://fluffynukeit.com/?page\\_id=535](http://fluffynukeit.com/?page_id=535) (with demo)

#### 7.13.5 arbtt

Report by:	Joachim Breitner
Status:	working

The program `arbtt`, the automatic rule-based time tracker, allows you to investigate how you spend your time, without having to manually specify what you are doing. `arbtt` records what windows are open and active, and provides you with a powerful rule-based language

to afterwards categorize your work. And it comes with documentation!

Since the last report, version 0.7 was released, sporting a progress bar while processing the data. Furthermore, Waldir Pimenta contributed a slick web site for the project.

### Further reading

- <http://arbtt.nomeata.de/>
- <http://www.joachim-breitner.de/blog/archives/336-The-Automatic-Rule-Based-Time-Tracker.html>
- [http://arbtt.nomeata.de/doc/users\\_guide/](http://arbtt.nomeata.de/doc/users_guide/)

#### 7.13.6 java-bridge

Report by:	Julian Fleischer
Status:	active development

The Java Bridge is a library for interfacing the Java Virtual Machine with Haskell code and vice versa. It comes with a rich DSL for discovering and invoking Java methods and allows to set up callbacks into the Haskell runtime. If exported via the FFI it is also possible to use Haskell libraries from within the JVM natively.

The package also offers a bindings generator which translates the API of a Java class or package into a Haskell API. Using the bindings generator it is possible to generate a Haskell module with a clean Haskell API that invokes Java behind the scenes. Typical conversions, for example byte arrays to lists or Java maps to lists of key value pairs, are taken care of. The generated bindings and predefined conversions are extensible by defining appropriate type class instances accordingly.

While the documentation for the bindings generator still needs improvement, the overall library is in a quite usable state.

The java bridge is published under the MIT license and available via hackage as `java-bridge`.

### Further reading

If you want to know more about the inner workings: The Java Bridge has been created as part of a bachelor thesis which you can access at <http://page.mi.fu-berlin.de/scravy/bridging-the-gap-between-haskell-and-java.pdf>.

## 8 Commercial Users

### 8.1 Well-Typed LLP

Report by:	Andres Löh
Participants:	Duncan Coutts

Well-Typed is a Haskell services company. We provide commercial support for Haskell as a development platform, including consulting services, training, and bespoke software development. For more information, please take a look at our website or drop us an e-mail at [info@well-typed.com](mailto:info@well-typed.com).

We are working for a variety of commercial clients, but naturally, only some of our projects are publically visible.

Well-Typed is now five years old! We've been sad to see Ian Lynagh, one of Well-Typed's founders, leaving the company this summer. We are grateful for all the hard work that Ian has done over the years.

We have hired two new consultants: Austin Seipp and Adam Gundry. Austin in particular is going to be taking over a lot of Ian's responsibilities in the development and maintenance of GHC (→ 3.1). We're currently busy preparing the GHC 7.8.1 release.

On behalf of the Industrial Haskell Group (IHG) (→ 8.3), we have been working on getting Hackage 2 (→ 6.3.1) ready for release for a while, and we're happy to say that the release has finally happened. Hackage 2 is live and works well since a couple of weeks now. Now is a good time to become involved with Hackage development, because it's now much easier to get changes merged and see them in production soon.

We continue to be involved in the community, maintaining several packages on Hackage and giving talks at a number of conferences. Some of our recent appearances are available online, such as Edsko's talk on Lazy I/O at Skills Matter, Andres's talk on Cloud Haskell at the Big Tech Day in Munich, and Andres's talk on Free Monads at the Haskell eXchange (links below).

We are continuing to offer training services. We offer regular courses in London and (new) in New York City, and on-demand on-site training courses elsewhere as well. For our London courses, the 2014 dates are now public, and the NYC dates will be known within the next few weeks.

We are of course always looking for new clients and projects, so if you have something we could help you with, just drop us an e-mail.

#### Further reading

- Company page: <http://www.well-typed.com>

- Blog: <http://blog.well-typed.com/>
- Training page: [http://www.well-typed.com/services\\_training](http://www.well-typed.com/services_training)
- Edsko's Lazy I/O talk: <http://skillsmatter.com/podcast/haskell/lazy-io-and-alternatives-in-haskell>
- Andres's Cloud Haskell talk: <http://www.techcast.com/events/bigtechday6/odeon-1345>
- Andres's Free Monads talk: <http://skillsmatter.com/podcast/haskell/monads-for-free>

### 8.2 Bluespec Tools for Design of Complex Chips and Hardware Accelerators

Report by:	Rishiyur Nikhil
Status:	commercial product

Bluespec, Inc. provides an industrial-strength language (BSV) and tools for high-level hardware design. Components designed with these are shipping in some commercial smartphones and tablets today.

BSV is used for all aspects of ASIC and FPGA design — specification, synthesis, modeling, and verification. All hardware behavior is expressed using *rewrite rules* (Guarded Atomic Actions). BSV borrows many ideas from Haskell — algebraic types, polymorphism, type classes (overloading), and higher-order functions. Strong static checking extends into correct expression of multiple clock domains, and to gated clocks for power management. BSV is universally applicable, from algorithmic “datapath” blocks to complex control blocks such as processors, DMAs, interconnects, and caches.

Bluespec's core tool synthesizes (compiles) BSV into high-quality Verilog, which can be further synthesized into netlists for ASICs and FPGAs using third-party tools. Atomic transactions enable design-by-refinement, where an initial executable approximate design is systematically transformed into a quality implementation by successively adding functionality and architectural detail. The synthesis tool is implemented in Haskell (well over 100K lines).

Bluesim is a fast simulation tool for BSV. There are extensive libraries and infrastructure to make it easy to build FPGA-based accelerators for compute-intensive software, including for the Xilinx XUPv6 board popular in universities, and the Convey HC-1 high performance computer.

BSV is also enabling the next generation of computer architecture education and research. Students implement and explore architectural models on FPGAs, whose speed permits evaluation using whole-system software.

## Status and availability

BSV tools, available since 2004, are in use by several major semiconductor and electronic equipment companies, and universities. The tools are free for academic teaching and research.

## Further reading

- *Abstraction in Hardware System Design*, R.S. Nikhil, in *Communications of the ACM*, 54:10, October 2011, pp. 36-44.
- *Bluespec, a General-Purpose Approach to High-Level Synthesis Based on Parallel Atomic Transactions*, R.S. Nikhil, in *High Level Synthesis: from Algorithm to Digital Circuit*, Philippe Coussy and Adam Morawiec (editors), Springer, 2008, pp. 129-146.
- *BSV by Example*, R.S. Nikhil and K. Czeck, 2010, book available on Amazon.com.
- <http://bluespec.com/SmallExamples/index.html>: from *BSV by Example*.
- <http://www.cl.cam.ac.uk/~swm11/examples/bluespec/>: Simon Moore's BSV examples (U. Cambridge).
- <http://csg.csail.mit.edu/6.375>: *Complex Digital Systems*, MIT courseware.
- <http://www.bluespec.com/products/BluDACu.htm>: A fun example with many functional programming features — BluDACu, a parameterized Bluespec hardware implementation of Sudoku.

## 8.3 Industrial Haskell Group

Report by:	Andres Löh
------------	------------

The Industrial Haskell Group (IHG) is an organization to support the needs of commercial users of Haskell.

The main activity of the IHG is to fund work on the Haskell development platform. It currently operates two schemes:

- The collaborative development scheme pools resources from full members in order to fund specific development projects to their mutual benefit.
- Associate and academic members contribute to a separate fund which is used for maintenance and development work that benefits the members and community in general.

In the past six months, the collaborative development scheme funded work on the new Hackage server (→ 6.3.1). Hackage 2 has now replaced the old Hackage and is in active use. The code of Hackage 2 is on Github. This is a good time to become involved with Hackage development, because it's now much easier to get changes merged and see them in production soon.

In general, details of the tasks undertaken are appearing on the Well-Typed (→ 8.1) blog, on the IHG status page and on standard communication channels such as the Haskell mailing list.

The collaborative development scheme is running continuously, so if you are interested in joining as a member, please get in touch. Details of the different membership options (full, associate, or academic) can be found on the website.

Two new associate members have just joined the IHG: Systor Vest and Erudify. There are now three full members, five associate members and one academic member in the IHG.

We are very interested in new members. If you are interested in joining the IHG, or if you just have any questions or comments, please drop us an e-mail at ([info@industry.haskell.org](mailto:info@industry.haskell.org)).

## Further reading

- <http://industry.haskell.org/>
- <http://industry.haskell.org/status/>

## 8.4 Barclays Capital

Report by:	Ben Moseley
------------	-------------

Barclays Capital has been using Haskell as the basis for our FPF (Functional Payout Framework) project for about seven years now. The project develops a DSL and associated tools for describing and processing exotic equity options. FPF is much more than just a payoff language — a major objective of the project is not just pricing but “zero-touch” management of the entire trade lifecycle through automated processing and analytic tools.

For the first half of its life the project focused only on the most exotic options — those which were too complicated for the legacy systems to handle. Over the past few years however, FPF has expanded to provide the trade representation and tooling for the vast majority of our equity exotics trades and with that the team has grown significantly in both size and geographical distribution. We now have eight permanent full-time Haskell developers spread between Hong Kong, Kiev and London (with the latter being the biggest development hub).

Our main front-end language is currently a deeply embedded DSL which has proved very successful, but we have recently been working on a new non-embedded implementation. This will allow us to bypass some of the traditional DSEL limitations (e.g., error messages and syntactical restrictions) whilst addressing some business areas which have historically been problematic. The new language is based heavily on arrows, but has a custom (restrictive but hopefully easier-to-use than raw arrow-notation) syntax. We are using a compiler from our custom DSL syntax into Haskell

source (with standard transformers from Ross Paterson’s “arrows” package) to provide the semantics for the language but plan to develop a number of independent backends. Our hope is that, over time, this will gradually replace our embedded DSL as the front end for all our tools. For the parsing part of this work we have been very impressed by Doaitse Swierstra’s `uiparsinglib` (→ 7.3.3).

We have been and remain very satisfied GHC users and feel that it would have been significantly harder to develop our systems in any other current language.

## 8.5 Oblomov Systems

Report by:

Martijn Schrage



Oblomov Systems is a one-person software company based in Utrecht, The Netherlands. Founded in 2009 for the Proxima 2.0 project (<http://www.haskell.org/communities/05-2010/html/report.html#sect6.4.5>), Oblomov has since then been working on a number of Haskell-related projects. The main focus lies on web-applications and (web-based) editors. Haskell has turned out to be extremely useful for implementing web servers that communicate with JavaScript clients or iPhone apps.

Awaiting the acceptance of Haskell by the world at large, Oblomov Systems also offers software solutions in Java, Objective C, and C#, as well as on the iPhone/iPad. Last year, Oblomov Systems has worked together with Ordina NV on a substantial Haskell project for the Council for the Judiciary in The Netherlands.

### Further reading

<http://www.oblomov.com>

## 8.6 OpenBrain Ltd.

Report by:

Tom Nielsen

OpenBrain Ltd. is developing a new platform for statistical computing that enables optimal decisions taking into account all the available information. We have developed a new statistical programming language (BAYSIG) that augments a Haskell-like functional programming language with Bayesian inference and first-class ordinary and stochastic differential equations. BAYSIG is designed to support a declarative style of programming where almost all the work consists in building probabilistic models of observed data. Data analysis, risk assessment, decision, hypothesis

testing and optimal control procedures are all derived mechanically from the definition of these models. We are targeting a range of application areas, including financial, clinical and life sciences data.

We are building a web application (<http://BayesHive.com>) to make this platform accessible to a wide range of users. Users can upload and analyse varied types of data using a point-and-click interface. Models and analyses are collected in literate programming-like documents that can be published by users as blogs.

We use Haskell for almost all aspects of implementing this platform. The BAYSIG compiler is written in Haskell, which is particularly well suited for implementing the recursive syntactical transformations underlying statistical inference. `BayesHive.com` is being developed in Yesod.

### Contact

[tomn@openbrain.org](mailto:tomn@openbrain.org)

### Further reading

<http://BayesHive.com>

## 9 Research and User Groups

### 9.1 Haskell at Eötvös Loránd University (ELTE), Budapest

Report by: PÁLI Gábor János  
Status: ongoing



#### Education

There are many different courses on Haskell and Agda that run at Eötvös Loránd University, Faculty of Informatics.

- Programming for first-year BSc students using Haskell, it is officially in the curriculum. It is also taught for foreign language students as part of their program.
- Advanced functional programming using Haskell, it is an optional course for BSc and MSc students.
- Programming in Agda as an optional course for BSc and MSc students.
- Other Haskell-related courses on Lambda Calculus, Type Theory and Implementation of Functional Languages.

There is an interactive online evaluation and testing system, called ActiveHs. It contains several hundred systematized exercises and it may be also used as a teaching aid. There is also some experimenting going on about supporting SVG graphics, and extending the embedded interpreter and testing environment with safe emulation of IO values, providing support for Agda.

We have been translating our course materials to English, some of the materials is already available.

#### Further reading

- Haskell course materials (in English): [http://pnyf.inf.elte.hu/fp/Overview\\_en.xml](http://pnyf.inf.elte.hu/fp/Overview_en.xml)
- Agda course materials (in English): <http://people.inf.elte.hu/divip/AgdaTutorial/Index.html> (→ 2.3)
- ActiveHs: <http://hackage.haskell.org/package/activehs>

### 9.2 Artificial Intelligence and Software Technology at Goethe-University Frankfurt

Report by: David Sabel  
Participants: Conrad Rau, Manfred Schmidt-Schauß

**Semantics of Functional Programming Languages.** Extended call-by-need lambda calculi model the semantics of Haskell. Our investigations of those calculi include correctness of strictness analysis using abstract reduction, equivalence of the call-by-name and call-by-need semantics, completeness of applicative bisimilarity w.r.t. contextual equivalence, and unsoundness of applicative bisimilarity in nondeterministic languages with `letrec`. We also have shown that any semantic investigation of Haskell should include the `seq`-operator, since extending the lazy lambda calculus by `seq` is not conservative, i.e. the semantics changes.

A recent result is that deciding (extended)  $\alpha$ -equivalence in languages with bindings (like `letrec`) is graph isomorphism complete. However, if the expressions are free of garbage (i.e. have no unused bindings) the problem can be solved efficiently.

**Concurrency.** We analyzed a higher-order functional language with concurrent threads, monadic IO, synchronizing variables and concurrent futures which models Concurrent Haskell. We proved correctness of program transformations, correctness of an abstract machine, and we have shown that this language conservatively extends the pure core language of Haskell, i.e. all program equivalences for the pure part also hold in the concurrent language. Most recently, we proved correctness of a highly concurrent implementation of Software Transactional Memory (STM) in a similar program calculus. We also proposed an alternative approach for implementing STM Haskell and prototypically implemented a library.

**Correctness of Program Transformations.** An ongoing project aims at automating correctness proofs of program transformations. To compute so-called forking and commuting diagrams we implemented a sound and complete algorithm as a combination of several unification algorithms in Haskell. To conclude the correctness proofs we automated the corresponding induction proofs (which use the diagrams) using automated termination provers for term rewriting systems.

**Grammar based compression.** This research topic focuses on algorithms on grammar compressed data like strings, matrices, trees, ... Our goal is to reconstruct known algorithms on uncompressed data (e.g. unification, matching, matrix operations, etc.) for their use on grammars without prior decompression.

We implemented several of those algorithms in Haskell.

### Further reading

<http://www.ki.informatik.uni-frankfurt.de/research/HCAR.html>

## 9.3 Functional Programming at the University of Kent

Report by: Olaf Chitil

The Functional Programming group at Kent is a subgroup of the Programming Languages and Systems Group of the School of Computing. We are a group of staff and students with shared interests in functional programming. While our work is not limited to Haskell, we use for example also Erlang and ML, Haskell provides a major focus and common language for teaching and research.

Our members pursue a variety of Haskell-related projects, several of which are reported in other sections of this report. Thomas Schilling passed his PhD viva on trace-based dynamic optimisations for Haskell programs. Three new PhD students joined the group this September. Stephen Adams is working on advanced refactoring of Haskell programs. Andreas Reuleaux is working on refactoring dependently typed functional programs. Maarten Faddegon is working on making tracing for Haskell practical and easy to use. Currently he is looking into combining ideas from the Haskell tracer Hat ( $\rightarrow$  6.4.4) and the Haskell object observation debugger Hood. Olaf Chitil refactored/reimplemented Hat to use standard Hackage libraries. Thus the code is simpler and more maintainable, but the functionality is unchanged. Scott Owens is working on verified compilers for the (strict) functional language CakeML.

We are always looking for more PhD students. We are particularly keen to recruit students interested in programming tools for verification, tracing, refactoring, type checking and any useful feedback for a programmer. The school and university have support for strong candidates: more details at <http://www.cs.kent.ac.uk/pg> or contact any of us individually by email.

We are also keen to attract researchers to Kent to work with us. There are many opportunities for research funding that could be taken up at Kent, as shown in the website <http://www.kent.ac.uk/researchservices/sciences/fellowships/index.html>. Please let us know if you're interested in applying for one of these, and we'll be happy to work with you on this.

Finally, if you would like to visit Kent, either to give a seminar if you're passing through London or the UK, or to stay for a longer period, please let us know.

### Further reading

- o PLAS group: <http://www.cs.kent.ac.uk/research/groups/plas/>
- o Haskell: the craft of functional programming: <http://www.haskellcraft.com>
- o Refactoring Functional Programs: <http://www.cs.kent.ac.uk/research/groups/plas/hare.html>
- o A trace-based just-in-time compiler for Haskell: <http://www.youtube.com/watch?v=PtEcls2t9Ws>
- o Scion, a library for building IDEs for Haskell: <http://code.google.com/p/scion-lib/>
- o Hat, the Haskell Tracer: <http://projects.haskell.org/hat/>
- o CakeML, a verification friendly dialect of SML: <https://cakeml.org>
- o Practical Lazy Typed Contracts for Haskell: <http://www.cs.kent.ac.uk/~oc/contracts.html>
- o Heat, an IDE for learning Haskell: <http://www.cs.kent.ac.uk/projects/heat/>

## 9.4 Formal Methods at DFKI and University Bremen and University Magdeburg

Report by: Christian Maeder  
Participants: Mihai Codescu, Christoph Lüth, Till Mossakowski  
Status: active development

The activities of our groups center on formal methods, covering a variety of formal languages and also translations and heterogeneous combinations of these.

We are using the Glasgow Haskell Compiler and many of its extensions to develop the Heterogeneous tool set (Hets). Hets is a parsing, static analysis and proof management tool incorporating various provers and different specification languages, thus providing a tool for heterogeneous specifications. Logic translations are first-class citizens.

The languages supported by Hets include the CASL family, such as the Common Algebraic Specification Language (CASL) itself (which provides many-sorted first-order logic with partiality, subsorting and induction), HasCASL, CoCASL, CspCASL, and an extended modal logic based on CASL. Other languages supported include propositional logic, QBF, Isabelle, Maude, VSE, TPTP, THF, FPL (logic of functional programs), LF type theory and still Haskell (via Programatica). More recently, ontology languages like OWL, RDF, Common Logic, and DOL (the Distributed Ontology Language) have been integrated.

Hets can speak to the following provers:

- o minisat, zChaff (SAT solvers),
- o SPASS, Vampire, Darwin, KRHyper and MathServe (automated first-order theorem provers),
- o Pellet and Fact++ (description logic tableau provers),

- Leo-II and Satallax (automated higher-order theorem provers),
- Isabelle (an interactive higher-order theorem prover),
- CSPCASL-prover (an Isabelle-based prover for Csp-CASL),
- VSE (an interactive prover for dynamic logic).

The user interface of the Hets implementation (about 200K lines of Haskell code) is based on some Haskell sources such as bindings to uDrawGraph (formerly Davinci) and Tcl/TK that we maintain and also gtk2hs ( $\rightarrow$  7.7.3). Additionally we have a command line interface and a prototypical web interface based on warp ( $\rightarrow$  5.2.2) with a RESTful API.

HasCASL is a general-purpose higher-order language which is in particular suited for the specification and development of functional programs; Hets also contains a translation from an executable HasCASL subset to Haskell. There is a prototypical translation of a subset of Haskell to Isabelle/HOL.

### Further reading

- Group activities overview:  
[http://www.informatik.uni-bremen.de/agbkb/forschung/formal\\_methods/](http://www.informatik.uni-bremen.de/agbkb/forschung/formal_methods/)
- CASL specification language:  
<http://www.cofi.info>
- distributed ontology language DOL:  
<http://www.ontoiop.org>
- Heterogeneous tool set:  
<http://hets.dfk.de>  
<http://www.informatik.uni-bremen.de/htk/>  
<http://www.informatik.uni-bremen.de/uDrawGraph/>

## 9.5 Haskell at Universiteit Gent, Belgium

Report by:	Tom Schrijvers
Participants:	Steven Keuchel

Haskell is one of the main research topics of the new Programming Languages Group at the Department of Applied Mathematics and Computer Science at the University of Ghent, Belgium.

**Teaching.** UGent is a great place for Haskell-aficionados:

- Make Haskell part of your studies with the elective course *Functional and Logic Programming Languages*.
- Explore the theory behind Haskell in the new master course on *Programming Language Fundamentals*.
- Explore Haskell in depth with one of our Haskell master thesis topics.
- Attend the thriving Ghent Functional Programming Group ( $\rightarrow$  9.10).

**Research.** Haskell-related projects of the group members and collaborators are:

- *Meta-Theory à la Carte*:

Formalizing meta-theory, or proofs about programming languages, in a proof assistant has many well-known benefits. However, the considerable effort involved in mechanizing proofs has prevented it from becoming standard practice. This cost can be amortized by reusing as much of an existing formalization as possible when building a new language or extending an existing one. Unfortunately reuse of components is typically ad-hoc, with the language designer cutting and pasting existing definitions and proofs, and expending considerable effort to patch up the results.

This work presents a more structured approach to the reuse of formalizations of programming language semantics through the composition of modular definitions and proofs. The key contribution is the development of an approach to induction for extensible Church encodings which uses a novel reinterpretation of the universal property of folds. These encodings provide the foundation for a framework, formalized in Coq, which uses type classes to automate the composition of proofs from modular components.

Several interesting language features, including binders and general recursion, illustrate the capabilities of our framework. We reuse these features to build fully mechanized definitions and proofs for a number of languages, including a version of mini-ML. Bounded induction enables proofs of properties for non-inductive semantic functions, and mediating type classes enable proof adaptation for more feature-rich languages.

This is joint work with Ben Delaware and Bruno Oliveira.

- *Generic Conversions of Abstract Syntax Representations*:

This work presents a datatype-generic approach to syntax with variable binding. A universe specifies the binding and scoping structure of object languages, including binders that bind multiple variables as well as sequential and recursive scoping. Two interpretations of the universe are given: one based on parametric higher-order abstract syntax and one on well-typed de Bruijn indices. The former provides convenient interfaces to embedded domain-specific languages, but is awkward to analyse and manipulate directly, while the latter is a convenient representation in implementations, but is unusable as a surface language. We show how to generically convert from the parametric HOAS interpretation to the de Bruijn interpretation thereby taking the pain from DSL developer to write the conversion themselves.

This is joint work with Johan Jeuring.



- *Modular Reasoning about Incremental Programming*: *Incremental Programming* (IP) is a programming style in which new program components are defined as increments of other components. Examples of IP mechanisms include: *Object-oriented programming* (OOP) inheritance, *aspect-oriented programming* (AOP) advice and *feature-oriented programming* (FOP). A characteristic of IP mechanisms is that, while individual components can be independently defined, the composition of components makes those components become tightly coupled, sharing both control and data flows. This makes reasoning about IP mechanisms a notoriously hard problem: *modular reasoning* about a component becomes very difficult; and it is very hard to tell if two tightly coupled components *interfere* with each other's control and data flows.

This work presents *modular reasoning about interference* (MRI), a *purely functional* model of IP embedded in Haskell. MRI models inheritance with mixins and side-effects with monads. It comes with a range of powerful reasoning techniques: equational reasoning, parametricity and reasoning with algebraic laws about effectful operations. These techniques enable modular reasoning about interference in the presence of side-effects.

MRI formally captures *harmlessness*, a hard-to-formalize notion in the interference literature, in two theorems. We prove these theorems with a non-trivial combination of all three reasoning techniques.

This is joint work with Bruno Oliveira and William Cook.

- *Search Combinators*:

Search heuristics often make all the difference between effectively solving a combinatorial problem and utter failure. Hence, the ability to swiftly design search heuristics that are tailored towards a problem domain is essential to performance improvement. In other words, this calls for a high-level domain-specific language (DSL).

The tough technical challenge we face when designing a DSL for search heuristics, is to bridge the gap between a conceptually simple specification language (high-level, purely functional and naturally compositional) and an efficient implementation (typically low-level, imperative and highly non-modular). We overcome this challenge with a systematic approach in Haskell that disentangles different primitive concepts into separate monadic modular mixin components, each of which corresponds to a feature in the high-level DSL. The great advantage of mixin components to provide a semantics for our DSL is its modular extensibility.

This is joint work with Guido Tack, Pieter Wuille, Horst Samulowitz and Peter Stuckey, following up on

*Monadic Constraint Programming*, a monadic DSL for Constraint Programming in Haskell.

### Further reading

- <http://users.ugent.be/~tschrijv/haskell.html>
- <http://users.ugent.be/~tschrijv/SearchCombinators/>
- <http://hackage.haskell.org/package/Monatron>
- <http://hackage.haskell.org/package/monadicccp>

## 9.6 Haskell in Romania

Report by:	Mihai Maruseac
Participants:	Dan Popa

In Romania, Haskell is taught at several universities across the country: in Bucharest at both University POLITEHNICA of Bucharest and University of Bucharest, in Bacău at “Vasile Alecsandri” University, in Braşov at “Transilvania” University, . . . However, everywhere the courses are only centered on the theoretical aspects of functional programming and (sometimes) type systems. As a result, very few students will use this language after the exam is taken.

However, small communities are created to promote the language. That was the case of the Ro/Haskell group from Bacău or FPBucharest group. Right now, almost all of these groups have stopped being active.

The main reason behind these failures is that the point of view in presenting the language is too deeply concerned with presenting its features and the purely functional aspect while hiding away the fact that you have to do some IO in real world applications. Basically, every activity of the previous groups and the subjects taught at universities regard Haskell only as a laboratory language.

A small group of people from Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, decided last year to change that. The new teachers and teaching assistants from the Programming Paradigm course organised the first “Functional Programming Summer School” in June 2012 where a few real-world topics were presented among more theoretical aspects.

This year, a small subgroup of the ROSEdu (<http://rosedu.org/>) community developed on the feedback from the summer school and created a plan towards making Haskell a known and usable language with a community around it. There were talks on Yesod and GHC at different events (OSOM, Talks by Softbinator) or companies (IXIA), some new projects were launched – some of them being turned into bachelor or masters diploma projects – and an workshop called “Programming Haskell from N00b to Real World Programmer” was organized in June, during ROSEdu Summer Workshops (<http://workshop.rosedu.org/2013/sesiuni/haskell>). At the end of the workshop the stu-

dents implemented IRC bots and Pacman-like games with a graphical interface and some dummy AI. Finally, some articles were published in the "Today Software Magazine" (<http://www.todaysoftmag.com/tsm/en/>) monthly magazine. This has prompted some Haskell-related hackathons at an byweekly Agile event called "Code and Beer".

But one of the major results of these activities is that the awareness of Haskell in Romanian communities has increased, leading to the launch of three small startup companies in Romanian towns.

## 9.7 fp-syd: Functional Programming in Sydney, Australia

Report by:	Erik de Castro Lopo
Participants:	Ben Lippmeier, Shane Stephens, and others

We are a seminar and social group for people in Sydney, Australia, interested in Functional Programming and related fields. Members of the group include users of Haskell, Ocaml, LISP, Scala, F#, Scheme and others. We have 10 meetings per year (Feb–Nov) and meet on the third (usually, sometimes fourth) Wednesday of each month. We regularly get 20–30 attendees, with a 70/30 industry/research split. Talks this year have included material on compilers, theorem proving, type systems, Haskell web programming, Haskell database libraries, Scala and the Free Monad. We usually have about 90 mins of talks, starting at 6:30pm, then go for drinks afterwards. All welcome.

### Further reading

- <http://groups.google.com/group/fp-syd>
- <http://fp-syd.ouroborus.net/>
- <http://fp-syd.ouroborus.net/wiki/Past/2013>

## 9.8 Functional Programming at Chalmers

Report by:	Jean-Philippe Bernardy
------------	------------------------

Functional Programming is an important component of the CSE department at Chalmers and University of Gothenburg. In particular, Haskell has a very important place, as it is used as the vehicle for teaching and numerous research projects. Besides functional programming, language technology, and in particular domain specific languages is a common aspect in our projects. We also hope to see all HCAR readers at ICFP 2014 in Gothenburg the first week of September! (Paper submissions due Saturday, 1 March 2014.)

**Property-based testing.** QuickCheck, developed at Chalmers, is one of the standard tools for testing Haskell programs. It has been ported to Erlang and

used by Ericsson, Quviq, and others. QuickCheck continues to be improved and tools and related techniques are developed:

- We have shown how to successfully apply QuickCheck to test polymorphic properties.
- A new exhaustive testing tool (testing-feat on Hackage) has been developed. It is especially suited to generate test cases from large groups of mutually recursive syntax tree types. A paper describing it was presented at the Haskell Symposium 2012.
- **Testing Type Class Laws:** the specification of a class in Haskell often starts with stating, in comments, the laws that should be satisfied by methods defined in instances of the class, followed by the type of the methods of the class. We have developed a library (ClassLaws) that supports testing such class laws using QuickCheck.

**Parsing: BNFC.** The BNF Converter (BNFC) is a frontend for various parser generators in various languages. BNFC is written in Haskell and is commonly used as a frontend for the Haskell tools Alex and Happy. BNFC has recently been extended in two directions:

- A Haskell backend, which offers incremental and parallel parsing capabilities, as well as the ability to parse context-free grammars in full generality, has been added to BNFC. The underlying concepts are described in a paper published at ICFP 2013.
- BNFC has been embedded in a library (called BNFC-meta on Hackage) using Template-Haskell. An important aspect of BNFC-meta is that it automatically provides quasi-quotes for the specified language. This includes a powerful and flexible facility for anti-quotation.

**Parsing: Combinators.** A new package for combinator-based parsing has been released on Hackage. The combinators are based on the paper *Parallel Parsing Processes*. The technique is based on parsing in parallel all the possibly valid alternatives. This means that the parser never “hold onto” old input. A try combinator is also superfluous.

**Parsing: Natural languages.** Grammatical Framework is a declarative language for describing natural language grammars. It is useful in various applications ranging from natural language generation, parsing and translation to software localization. The framework provides a library of large coverage grammars for currently fifteen languages from which the developers could derive smaller grammars specific for the semantics of a particular application.

**Generic Programming.** Starting with Polytypic Programming in 1995 there is a long history of generic programming research at Chalmers. Recent developments include fundamental work on [parametricity](#). This work has led to the development of a new kind of abstraction, to generalize notions of erasure. This means that a new kind of generic programming is available to the programmer. A paper describing the idea was presented in [ICFP 2013](#).

Our research on generic-programming is lively, as witnessed by a constant stream of publications: [Testing Type Class Laws](#), [Functional Enumeration of Algebraic Types \(FEAT\)](#), [Testing versus proving in climate impact research](#) and [Dependently-typed programming in scientific computing — examples from economic modelling](#). The last two are part of our effort to contribute to the emerging research programme in [Global Systems Science](#).

**Program Inversion/bidirectionalization.** Program transformation systems that generate pairs of programs that are some sort of inverses of each other. The pairs are guaranteed to be consistent by construction with respect to certain laws. Applications include pretty-printing/parsing ([→ 7.3.1](#)), XML transformation etc. The work is done in collaboration with University of Tokyo and University of Bonn.

**Language-based security.** [SecLib](#) is a light-weight library to provide security policies for Haskell programs. The library provides means to preserve confidentiality of data (i.e., secret information is not leaked) as well as the ability to express intended releases of information known as declassification. Besides confidentiality policies, the library also supports another important aspect of security: integrity of data. [SecLib](#) provides an attractive, intuitive, and simple setting to explore the security policies needed by real programs.

**Type theory.** Type theory is strongly connected to functional programming research. Many dependently-typed programming languages and type-based proof assistants have been developed at Chalmers. The [Agda](#) system ([→ 4.1](#)) is the latest in this line, and is of particular interest to Haskell programmers. We encourage you to experiment with programs and proofs in [Agda](#) as a “dependently typed Haskell”.

**Embedded domain-specific languages.** The functional programming group has developed several different domain-specific languages embedded in Haskell. The active ones are:

- **Feldspar** ([→ 7.12.1](#)) is a domain-specific language for digital signal processing (DSP), developed in co-operation by Ericsson, Chalmers FP group and Eötvös Loránd (ELTE) University in Budapest.

- **Obsidian** is a language for data-parallel programming targeting GPUs.

Most recently we used [Obsidian](#) to implement an interesting variation of counting sort that also removes duplicate elements. This work was presented at [FHPC 2013](#).

We are also working on general methods for [EDSL](#) development:

- **Syntactic** is a library that aims to support the definition of [EDSLs](#). The core of the library was presented at [ICFP 2012](#). The paper presents a generic model of typed abstract syntax trees in Haskell, which can serve as a basis for a library supporting the implementation of deeply embedded DSLs.

- **Names For Free.** A new technique for representing names and bindings of object languages represented as Haskell data types has been developed. The essence of the technique is to represent names using *typed* de Bruijn indices. The type captures exactly the context where the index is valid, and hence is as safe to use as a name. The technique was presented at [Haskell Symposium 2013](#).

- **Circular Higher-Order Syntax** We have also developed a light-weight method for generating names while building an expression with binders. The method lends itself to be used in the front end of [EDSLs](#) based on higher-order syntax. The technique was presented at [ICFP 2013](#).

- **Simple and Compositional Monad Reification** A method for reification of monads (compilation of monadic embedded languages) that is both simple and composable. The method was presented at [ICFP 2013](#).

**Automated reasoning.** We are responsible for a suite of automated-reasoning tools:

- **Equinox** is an automated theorem prover for pure first-order logic with equality. [Equinox](#) actually implements a hierarchy of logics, realized as a stack of theorem provers that use abstraction refinement to talk with each other. In the bottom sits an efficient SAT solver. [Paradox](#) is a finite-domain model finder for pure first-order logic with equality. [Paradox](#) is a MACE-style model finder, which means that it translates a first-order problem into a sequence of SAT problems, which are solved by a SAT solver.

- **Infinox** is an automated tool for analysing first-order logic problems, aimed at showing finite unsatisfiability, i.e., the absence of models with finite domains. All three tools are developed in Haskell.

- **QuickSpec** generates algebraic specifications for an API automatically, in the form of equations verified by random testing. <http://www.cse.chalmers.se/~nicsma/quickspec.pdf>
- **Hip** (the Haskell Inductive Prover) is a new tool to automatically prove properties about Haskell programs by using induction or co-induction. The approach taken is to compile Haskell programs to first order theories. Induction is applied on the meta level, and proof search is carried out by automated theorem provers for first order logic with equality.
- On top of Hip we built **HipSpec**, which automatically tries to find appropriate background lemmas for properties where only doing induction is too weak. It uses the translation and structural induction from Hip. The background lemmas are from the equational theories built by QuickSpec. Both the user-stated properties and those from QuickSpec are now tried to be proven with induction. Conjectures proved to be theorems are added to the theory as lemmas, to aid proving later properties which may require them. For more information, see <http://web.student.chalmers.se/~danr/hipspec-atx.pdf> the draft paper.

**Teaching.** Haskell is present in the curriculum as early as the first year of the BSc programme. We have four courses solely dedicated to functional programming (of which three are MSc-level courses), but we also provide courses which use Haskell for teaching other aspects of computer science, such the syntax and semantics of programming languages, compiler construction, data structures and parallel programming.

## 9.9 Functional Programming at KU

Report by:	Andy Gill
Status:	ongoing



Functional Programming continues at KU and the Computer Systems Design Laboratory in ITTC! The System Level Design Group (lead by Perry Alexander) and the Functional Programming Group (lead by Andy Gill) together form the core functional programming initiative at KU. There are three major Haskell projects at KU (as well as numerous smaller ones): the GHC rewrite plugin HERMIT (→ 7.3.5), the VHDL

generator Kansas Lava (→ 7.12.2) and the JavaScript generator Sunroof (→ 5.2.8).

Nicolas Frisby, who defended his PhD from KU last summer, spent the spring at MSR Cambridge, working on optimizations inside the Glasgow Haskell compiler. Andrew Farmer spent the spring in Portland, OR, working with functional programmers at Portland State.

### Further reading

- The Functional Programming Group: <http://www.ittc.ku.edu/csdl/fpg>
- CSDL website: [https://wiki.ittc.ku.edu/csdl/Main\\_Page](https://wiki.ittc.ku.edu/csdl/Main_Page)

## 9.10 Ghent Functional Programming Group

Report by:	Andy Georges
Participants:	Jeroen Janssen, Tom Schrijvers, Jasper Van der Jeugt
Status:	active

The Ghent Functional Programming Group is a user group aiming to bring together programmers, academics, and others interested in functional programming located in the area of Ghent, Belgium. Our goal is to have regular meetings with talks on functional programming, organize functional programming related events such as hackathons, and to promote functional programming in Ghent by giving after-hours tutorials. While we are open to all functional languages, quite frequently, the focus is on Haskell, since most attendees are familiar with this language. The group has been active for two and a half years, holding meetings on a regular basis.

We have reported in previous HCARs on the first eleven meetings. Since May 2012, we had a single meeting. The GhentFPG #12 meeting took place on May 8, 2012 and involved two talks.

- Tom Schrijvers — Discussion on the Flemish Programming Contest 2012, with a focus on using the right Haskell data types for solving several of the given problems.
- Jasper Van der Jeugt — Tutorial on parallelisation in Haskell.

The attendance at the meetings usually varies between 10 to 15 people. We do have a number of Ghent University students attending. However, due to a shift in venue, the attendance has dropped slightly.

The plans for the fall 2012 Hackathon have shifted due to busy schedules of the GhentFPG organisers. In this academic year, we do plan to review the approach used during the meetings, because talks seem to attract more attendees compared to problem solving or coding events.

If you want more information on GhentFPG you can follow us on twitter (@ghentfpg), via Google Groups (<http://groups.google.com/group/ghent-fpg>), or by visiting us at <irc.freenode.net> in channel #ghentfpg.

### Further reading

- [http://www.haskell.org/haskellwiki/Ghent\\_Functional\\_Programming\\_Group](http://www.haskell.org/haskellwiki/Ghent_Functional_Programming_Group)
- <http://groups.google.com/group/ghent-fpg>

## 9.11 Odessa Haskell User Group

Report by: Roman Cheplyaka

Odessa Haskell User Group was founded in May 2012. Since then we met every month at a cafe to talk about Haskell and share the latest news.

In May 2013 with help from Provectus IT (<http://www.provectus-it.com/>) we organized OdHac, the first big European Haskell hackathon in Eastern Europe. It gathered 27 Haskellers from Ukraine, Russia, England, Switzerland, Netherlands, Hungary and Belgium.

In October 2013 we had our first meeting with a talk, hosted by DataArt (<http://dataart.ua/>). It was devoted to the Haskell ecosystem.

Now we plan to hold such meetings with a talk every month. If you live in Odessa or are just visiting, it would be our pleasure to have you as a speaker or guest.

Announcements about our future meetings are published at <http://odhug.github.io/> (in Russian). Also feel free to get in touch with the organizer directly (<http://ro-che.info/>).

We offer different courses using Haskell. Our Summer School on Applied Functional Programming has been running successfully for several years now. After a first course in Haskell in second year of our BSc program, there are several other courses using Haskell that we offer our students, including courses on parsing, compiler construction, advanced functional programming, dependently typed programming and generic programming.

### Further reading

- <http://www.cs.uu.nl/wiki/Center>
- <http://www.cs.uu.nl/wiki/USCS>

## 9.12 Functional Programming at Utrecht University

Report by: Wouter Swierstra, Jeroen Bransen  
Participants: ST Group of Utrecht University

Functional programming is an important part of the Software Technology Group at Utrecht University, Utrecht, The Netherlands. We use Haskell for numerous research projects including:

- The Utrecht Haskell Compiler (→ 3.3);
- The Utrecht University Attribute Grammar Compiler (→ 5.3.2);
- Program analysis and type error diagnosis, for example for plagiarism detection (→ 7.2.3);
- The Ask-Elle programming tutor and other interactive learning environments (→ 7.2.4);
- and various libraries for generic programming, such as multirec and regular.