Haskell Communities and Activities Report

http://tinyurl.com/haskcar

Twenty-First Edition — November 2011

Janis Voigtländer (ed.)

Andreas Abel
Dmitry Astapov
Justin Bailey
Mario Blažević
Björn Buckwalter
Douglas Burke
Roman Cheplyaka
Nils Anders Danielsson
Daniel Díaz
Patai Gergely
Andy Gill
Marco Gontijo
Sönke Hahn
PÁLI Gábor János
Oleg Kiselyov
Ben Lippmeier
Tom Lokhorst
Christian Maeder
Alex McLean
Simon Michael
Dino Morelli
Takayuki Muranushi
Thomas van Noort
Jens Petersen
Antonio M. Quispe
Martijn Schrage
Axel Simon
Michael Snoyman
Doaitse Swierstra
Sergei Trofimovich
Janis Voigtländer
Kazu Yamamoto

Iain Alexander
Emil Axelsson
Doug Beardsley
Gwern Branwen
Bryan Buecking
Carlos Camarão
Olaf Chitil
James Deng
Atze Dijkstra
Jürgen Giesl
George Giorgidze
Torsten Grust
Malte Harder
Jeroen Janssen
Michal Konečný
Andres Löh
Rita Loogen
José Pedro Magalhães
Vivian McPhail
Arie Middelkoop
JP Moresmau
Jürgen Nicklisch-Franken
Johan Nordlander
Simon Peyton Jones
David Sabel
Tom Schrijvers
Ganesh Sittampalam
Andy Stewart
Henning Thielemann
Thomas Tuegel
David Waern
Brent Yorgey

Heinrich Apfelmus
Christiaan Baaij
Jean-Philippe Bernardy
Joachim Breitner
Joel Burget
Erik de Castro Lopo
Duncan Coutts
Dominique Devriese
Facundo Dominguez
Brett G. Giles
Dmitry Golubovsky
Jurriaan Hage
Bastiaan Heeren
Csaba Hruska
Eric Kow
Hans-Wolfgang Loidl
John MacFarlane
Ketil Malde
Alp Mestanogullari
Neil Mitchell
Ben Moseley
Rishiyur Nikhil
David M. Peixotto
Dan Popa
Uwe Schmidt
Jeremy Shaw
Jan Šnajder
Martin Sulzmann
Simon Thompson
Marcos Viera
Greg Weber

Preface

This is the 21st edition of the Haskell Communities and Activities Report. As usual, fresh entries are formatted using a blue background, while updated entries have a header with a blue background. Entries for which I received a liveness ping, but which have seen no essential update for a while, have been replaced with online pointers to previous versions. Other 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 call for new entries and updates to existing ones will be issued on the usual 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, ⟨hcar@haskell.org⟩

Contents

1 Community

1.1 haskell.org

Report by:	Ganesh Sittampalam
Participants:	Jason Dagit, Ian Lynagh, Don Stewart, Johan Tibell, Vo Minh Thu, Malcolm Wallace, Edward Z. Yang
Status:	active

The haskell.org committee was formed a year ago to formalise the previously ad-hoc arrangements around managing the haskell.org infrastructure and money. The committee's "home page" is at http://www.haskell.org/haskellwiki/Haskell.org_committee, and occasional publicity is via a blog (http://haskellorg.wordpress.com) and twitter account (http://twitter.com/#!/haskellorg) as well as the Haskell mailing list.

In our first year of operation, the following has happened:

haskell.org incorporation

The most important work for the year has been trying to get the ownership of haskell.org resources — principally some money from our GSoC participation, and various machines — on a sounder footing.

At the moment, Galois is kindly holding funds on behalf of haskell.org. However, this causes them administrative difficulties and it would also be better for haskell.org for them to be held separately in a vehicle with tax-free status (at least in the US) that can also accept donations.

The main option we have been exploring is joining the Software Freedom Conservancy (http://www.sfconservancy.org). After seeking the community's consent, we have contacted them to begin the application process. Unfortunately they are currently rather overworked and as they prioritise work for existing projects over accepting new ones, we do not yet know when there will be progress with this.

In the meantime we are also investigating joining an alternative, Software in the Public Interest (http://www.spi-inc.org). Discussions about this option are still ongoing.

The committee would like to thank Jason Dagit who has been helping us to make progress on this issue over the last few months, with the support of his employer Galois.

Subdomain policy

In response to various requests for subdomains of haskell.org, we have formulated the following policy, now (belatedly!) documented at http://www.haskell.org/haskellwiki/Haskell.org_domain#Policy_on_adding_new_subdomains

*Subdomains should be used for **services** rather than **content**.*

Content should normally be hosted at subpaths of http://www.haskell.org

So for example a Haskell graphics related website should normally go at http://www.haskell.org/graphics, rather than http://graphics.haskell.org.

In contrast, during the year, we did add revdeps.hackage.haskell.org for a hackage reverse-dependency lookup service, and of course hackage.haskell.org already exists.

Clearly the line between services and content, and indeed the precise definitions of each, is something of a grey area, and we are certainly happy to be flexible particularly if there are technical or other reasons for doing things one way. Our overall goal is to minimise unnecessary proliferation of subdomains and to try to keep the haskell.org domain reasonably well organised, while still helping people do useful things with it.

Move of www.haskell.org to a new dedicated host

For many years, www.haskell.org was generously hosted by Paul Hudak at Yale. This was becoming increasingly expensive for him so in late 2010 we moved to a new dedicated host (lambda.haskell.org). At the same time we put in place a policy that lambda would host only "meta" community resources, thus limiting the number of people who need to have accounts on it. For some time before this new project content had been created on community.haskell.org anyway, and this move gave us the opportunity to move "legacy" sites such as Gtk2Hs over to community. In addition, community.haskell.org is now also a VM running on the same machine.

The committee as a whole's involvement in this was only to approve the change — the sysadmin team did all the actual work.

General

The haskell.org infrastructure as a whole is still in a rather tenuous state. While the extreme unreliability we saw for a while has improved with the reorganisation, the level of sysadmin resource/involvement is still inadequate. The committee is open to ideas on how to improve the situation.

Unfortunately we cannot provide a full statement of haskell.org's accounts with this report; we are doing our best to track down the necessary information and will produce them as soon as possible. Better control and visibility of our finances and assets is of course one of the benefits we are seeking by affiliating with SFC or SPI.

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 November 2010 HCAR, Haskellers has added job postings, strike forces, and the ever important bling, as well as a brand new, community-developed site design. Haskellers is quickly approaching 800 active accounts. To be clear, the site is 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 Haskell: the craft of functional programming, 3rd edition

Report by: Simon Thompson

The third edition of one of the leading textbooks for beginning Haskell programmers is thoroughly revised throughout. New material includes thorough coverage of property-based testing using QuickCheck and an additional chapter on domain-specific languages as well as a variety of new examples and case studies, including simple games.

Existing material has been expanded and re-ordered, so that some concepts — such as simple data types and input/output — are presented at an earlier stage. The running example of Pictures is now implemented using web browser graphics as well as lists of strings.

The book uses GHCi, the interactive version of the Glasgow Haskell Compiler, as its implementation of choice. It has also been revised to include material about the Haskell Platform, and the Hackage online database of Haskell libraries. In particular, readers are given detailed guidance about how to find their way around what is available in these systems.

Publication details:

◦ Published by Addison Wesley, 2011. ISBN 0201882957.

Book website:

◦ http://www.haskellcraft.com

Solutions for *bona fide* instructors are available from the Pearson website http://www.pearsoned.co.uk/HigherEducation/Booksby/Thompson/

2.2 The Monad.Reader

Report by: Brent Yorgey

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 have been two new issues. Issue 18, published in July 2011, featured articles on a monadic formulation of MapReduce, parallel monad comprehensions, and attributed variables. Issue 19, published in October 2011, was a special issue on parallelism and concurrency, featuring an article on the Mighttpd web server, a tutorial on the use of MPI from Haskell, and an article about pipelines of coroutine-based processes.

Further reading

http://themonadreader.wordpress.com/

2.3 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 two additions:

Type-safe Formatted IO

A set of several articles describes various type-safe implementations of printf and scanf, with the same format descriptor. A type-safe printf converts the sequence of heterogeneous arguments to a string according to a given format descriptor; the number and the types of the arguments must agree with the descriptor. Haskell's Text.Printf.printf is not type-safe by the above definition since the type checker does not stop the programmer from passing to Text.Printf.printf more or fewer arguments than required by the formatting string. The dual type-safe scanf extracts a sequence of heterogeneous arguments from a string by interpreting the same format descriptor as a heterogeneous sequence of patterns binding zero or more variables. Although type-safe printf received a lot of attention (from Danvy, Hinze, Asai), the type-safe scanf is often neglected. Apparently it has been unknown if type-safe printf and scanf could share the same format descriptor.

Our implementations of type-safe printf and scanf all share the same insight of a simple embedded domain-specific language (DSL) of formatting patterns. The functions printf and scanf are two interpreters of the language, building or parsing a string according to the given pattern. The format descriptor, a term in our DSL, can be interpreted in far more than two ways, producing a family of printf/scanf-like functions.

The DSL of formatting patterns can be embedded into Haskell as a (generalized) algebraic data type, or as a family of overloaded functions (a type class). To the end user, the difference is hardly noticeable. However, whereas the first embedding requires GADT, the second one is entirely in Haskell98 and is extensible.

Finally, we implement printf that takes a C-like format string and the variable number of other arguments. Unlike C of Haskell's printf, ours is total: if the types or the number of the other arguments do not match the format string, a type error is reported. Likewise, we build a type-safe scanf that takes a C-like format string and the poly-variadic consumer function. We use Template Haskell to translate the format string to a term in the DSL of format descriptors.

http://okmij.org/ftp/type-formatting/

Annotating trees post factum

A common problem is annotating nodes of an already constructed tree or other such data structure with arbitrary new data. The original tree had been defined with no provision for node attributes, and we are not at liberty to change the data type definition. We should not even require rebuilding of the tree as we add annotations to its nodes. Our code must be pure functional; in particular, the tree to annotate should remain as it was. Finally, our solution should be expressible in a typed language without resorting to the Universal type.

Tree annotations are common in compilers, associating each abstract syntax tree node with source location data, inferred type, results of various analyses.

Our solution relies on the observation that each node in a finite tree can be identified by its path – a sequence of integers – with a decidable identity and order. To annotate tree nodes we build a *separate* finite map data structure associating nodes' paths with their annotations. To add annotations of a different type, we build another map.

http://okmij.org/ftp/Algorithms.html#tree-annot

2.4 A Tutorial on the Enumerator Library

Report by:	Kazu Yamamoto

Enumerator/Iteratee (EI) developed by Oleg Kiselyov is an API to enable modular programming in the IO monad. A popular implementation of EI is the *enumerator* library developed by John Millikin. This tutorial is a gentle introduction of the background of EI and how to use the *enumerator* library.

Further reading

http://www.mew.org/~kazu/proj/enumerator/

2.5 Practice of Functional Programming

Report by:	Dmitry Astapov
Status:	seven issues out, issue #8 is looming ahead, collecting materials for more

"Practice of Functional Programing" is a Russian electronic magazine promoting functional programming. The magazine features articles that cover both theoretical and practical aspects of the craft. Significant amount of the already published material is directly related to Haskell.

The magazine attempts to keep a bi-monthly release schedule, with Issue #7 leaving the press at the end of April 2011. Full contents of current and past issues are available in PDF from the official site of the magazine free of charge. Articles are in Russian, with English annotations.

Further reading

http://fprog.ru/ for issues ##1–7

3 Implementations

3.1 Haskell Platform

Report by: Duncan Coutts

Background

The Haskell Platform (HP) is the name of the "blessed" set of libraries and tools on which to build further Haskell libraries and applications. It takes a core selection of packages from the more than 3500 on Hackage (→ 6.8.1). It is intended to provide a comprehensive, stable, and quality tested base for Haskell projects to work from.

Historically, GHC shipped with a collection of packages under the name `extralibs`. Since GHC 6.12 the task of shipping an entire platform has been transferred to the Haskell Platform.

Recent progress

There has not been a release in the last 6 months. While the plan calls for major releases every 6 months this has not happened for a number of reasons. We took the decision not to base a major release on GHC-7.2.1 and no new release in the 7.2.x series is expected. We ran into some problems trying to prepare a release using GHC-7.0.4, however we may yet do a release using GHC-7.0.4.

Looking forward

Major releases are supposed to take place on a 6 month cycle. There will be a major release in Spring 2012 which will be based on the GHC-7.4.x series.

Our systems for coordinating and testing new releases remains too time consuming, involving too much manual work. Help from the community on this issue would be very valuable.

The platform steering committee will be proposing some modifications to the community review process for accepting new packages into the platform process with the aim of reducing the burden for package authors and keeping the review discussions productive. Though we will be making some modifications, we would still like to invite package authors to propose new packages. This can be initiated at any time. We also invite the rest of the community to take part in the review process on the libraries mailing list ⟨libraries@haskell.org⟩. The procedure involves writing a package proposal and discussing it on the mailing list with the aim of reaching a consensus. Details of the procedure are on the development wiki.

Further reading

http://haskell.org/haskellwiki/Haskell_Platform
- Download: http://hackage.haskell.org/platform/
- Wiki: http://trac.haskell.org/haskell-platform/
- Adding packages: http://trac.haskell.org/haskell-platform/wiki/AddingPackages

3.2 The Glasgow Haskell Compiler

Report by: Simon Peyton Jones
Participants: many others

GHC is still humming along, with the 7.2.1 release (more of a "technology preview" than a stable release) having been made in August, and attention now focused on the upcoming 7.4 branch. By the time you read this, the 7.4 branch will have been created, and will be in "feature freeze". We will then be trying to fix as many bugs as possible before releasing later in the year.

We advertised 7.2 as a technology preview, expecting 7.4 to merely consolidate the substantial new features in 7.2. But as it turns out GHC 7.4 will have a further wave of new features, especially in the type system. Significant changes planned for the 7.4 branch are:

Declarations at the GHCi prompt. Daniel Winograd-Cort (with help from Simon Marlow) has extended GHCi so that it is possible to give any declaration at the ghci prompt. For example,

```
Prelude> data D = D Int
Prelude> case D 5 of D x -> print x
5
```

This has already been merged, so will definitely be in 7.4.

Data type promotion and kind polymorphism. As we do more and more type-level programming, the lack of a decent kind system (to make sure that your type-level programs make sense) has become an increasingly pressing issue. If all goes well, GHC 7.4 will take a substantial step forward:

- First, all simple data types (including lists and tuples) will automatically be "promoted" to be kinds as well, a design inspired by Conor McBride's Strachclyde Haskell Extension [SHE]. For example:

  ```
  type family F :: Bool -> *
  type instance F True  = Int
  type instance F False = Char
  ```

- Second, GHC will support full kind polymorphism. For example, consider the following data type declaration

  ```
  data T f a = MkT (f a)
  -- T :: forall k. (k -> *) -> k -> *
  ```

  GHC will now infer the polymorphic kind signature above, rather that "defaulting" to `T :: (*->*) -> * -> *` as Haskell98 does.

These new kind-system developments are described in "Giving Haskell a promotion" [KindPolymorphism]. Julien Cretin and José Pedro Magalhães have done all the implementation.

Constraint kinds. Max Bolingbroke has implemented another extension to GHC's kind system, by adding the kind `Constraint` that classifies type constraints. This turns out to be a rather neat way to implement all the joy of Tom Schrijvers and Dominic Orchard's paper "Haskell type constraints unleashed" [Unleashed]. For example, you can now say

```
type Stringy a = (Show a, Read a)
f :: Stringy a => a -> a
f = read . show
```

Here, the constraint `(Stringy a)` is a synonym for `(Show a, Read a)`. More importantly, by combining with associated types, we can write some fundamentally new kinds of programs:

```
class Coll c where
  type X a :: Constraint
  insert :: X a => a -> c a -> c a
instance Coll [] where
  type X a = Eq a
  insert x []  = [x]
  insert x ys0@(y:ys)
    | x==y      = ys0
    | otherwise = y : insert x ys
```

Here `X` is an associated constraint synonym of the class `Coll`. The key point is that different instances can give different definitions to `X`. The GHC wiki page describes the design [WikiConstraint], and Max's blog posts give more examples [ConstraintFamlies, ConstraintKind].

Associated type synonym defaults. Haskell lets you give a default method for the operations of a class. Associated type synonym defaults let you declare a default type instance for the associated type synonyms of a class. This feature, implemented by Max Bolingbroke, nicely fills out a missing design corner. For example

```
class C a where
  type T a
  type T a = [a]  -- Default synonym
  f :: T a -> a
instance C Int
  f (x:xs) = x    -- No definition
                  -- given for T
```

Since we do not give a definition for `T` in the instance declaration, it filled in with the default given in the class declaration, just as if you had written `type T Int = [Int]`.

Monad comprehensions. After a long absence, monad comprehensions are back, thanks to George Giorgidze and his colleagues. With `{-# LANGUAGE MonadComprehensions #-}` the comprehension `[ f x | x <- xs, x>4 ]` is interpreted in an arbitrary monad, rather than being restricted to lists. Not only that, it also generalises nicely for parallel/zip and SQL-like comprehensions. The aforementioned generalisations can be turned on by enabling the `MonadComprehensions` extension in conjunction with the `ParallelListComp` and `TransformListComp` extensions.

Rebindable syntax is fully supported for standard monad comprehensions with generators and filters. We also plan to allow rebinding of the parallel/zip and SQL-like monad comprehension notations.

For further details and usage examples, see the paper "Bringing back monad comprehensions" [MonadComp] at the 2011 Haskell Symposium.

Improved implementation of type constraints. Over the last six months, Dimitrios and Simon PJ (with Stephanie Weirich and Brent Yorgey) have figured out several improvements to the GHC's type constraint solver and its strongly-typed Core language. The changes to the constraint solver eliminate hundreds of lines of code, and make it more efficient as well. The changes to the Core language make it treat equality constraints uniformly with other type constraints; this makes the internals vastly more uniform. These changes are mostly invisible to programmers, but the changes to Core allow us to support equality superclasses for the first time. Details in our paper "Practical aspects of evidence-based compilation in System FC" [NewFC]

Profiling and hpc overhaul. GHC currently has three different ways of tracking which pieces of code are executed: cost-centre profiling, HPC coverage, and GHCi debugger breakpoints. Each is implemented in a different, and somewhat ad hoc way. Simon Marlow has overhauled the whole system, unifiying the three mechanisms into one. On the way he has improved the semantics of cost centre stacks, which should lead to more useful time and space profiles. The `+RTS -xc` runtime flag, which displays a stack backtrace when an exception is thrown, has been greatly improved and should now produce useful information in most cases (it is still only available when the program is compiled for profiling, however).

Changes to the way Safe Haskell works [SafeHaskell]. David Terei has improved the design of Safe Haskell since the 7.2.1 release. In particular, it will no longer cause build failures for users who do not explicitly enable it. The checking that a package is trusted will only be done now if the `-fpackage-trust` flag is present. This allows package authors to use the `Trustworthy` pragma as they please and not worry that a users local package configuration will cause build failures. Users who are explicitly using Safe Haskell to construct secure systems should make use of the `-fpackage-trust` flag to maintain the security of the old design. Also since the 7.2.1 release, the safe status of a module will now be automatically inferred by Safe Haskell. These two changes make Safe Haskell easier to use and push it behind the scenes where it mostly belongs.

Joining in

We continue to receive some fantastic help from a number of members from the Haskell community. Amongst those who have rolled up their sleeves recently are:

- Ben Gamari, Karel Gardas and Stephen Blackheath have been working towards getting a registerised ARM port working.
- Many people, including Sergei Trofimovich, Erik de Castro Lopo, Joachim Breitner, Thorkil Naur, David M Peixotto and Ben Lippmeier, have contributed platform specific fixes for other platforms.
- Reiner Pope added Template Haskell support for unresolved infix expressions and patterns.
- José Pedro Magalhães has replaced the old generics support with a new design.
- Peter Wortmann taught GHC how to compile Objective-C++ files.
- Sam Anklesaria added support for additional .ghci files.
- Mikolaj Konarski and Duncan Coutts have improved GHC's event logging.
- Geoffrey Mainland improved error messages for unterminated quasiquotations.
- Johan Tibell implemented a "population count" primitive, and some other optimisations.
- Ross Paterson has fixed some problems with Arrows.
- Edward Z. Yang has been improving the RTS.
- George Roldugin improved the sync-all tool used by GHC developers.
- Austin Seipp has been improving some of the compiler documentation.
- Miscellaneous fixes and improvements from Daniel Fischer, Michal Terepeta and Lennart Kolmodin.

As ever, there is a lot still to do, and if you wait for us to do something then you may have to wait a long time. So do not wait; join in!

Other developments

Work continues on improving GHC in various directions. Active projects we know about include:

Cloud Haskell. The first version of Cloud Haskell has been released, aiming to bring Erlang-style distributed actors to Haskell (http://hackage.haskell.org/package/remote). See also the paper at Haskell Symposium 2011 [CloudHaskell]. Next, we are working on expanding the backend to work with HPC environments.

Parallel GHC project. Microsoft Research is funding a 2-year project to push the real-world use of parallel Haskell. We are now into the second year of the project and have some promising results from the project partners. We have also taken on two new commercial partner organisations which are interested in using Cloud Haskell. In addition to helping the project partners we have been working on parallel profiling tools: we made a release of [ThreadScope] with new features for profiling programs that use par sparks. For more details, see the HCAR Parallel GHC project entry (→ 5.1.3), and the project home page [ParallelGhcProject]

Data Parallel Haskell. GHC 7.2 includes rudimentary support for Data Parallel Haskell — just enough for a little experimentation and to run simple benchmarks. We are working on significantly improving this for GHC 7.4. In particular, we aim to support the use of basic types and classes from the standard Prelude (replacing the minimalistic mock Prelude that DPH programs had to use so far), and we are working on drastically improved space and time complexity for shared data structures in nested parallel programs, such as the Barnes-Hut n-body algorithm.

Binary distributions of GHC 7.x require the installation of separate Data Parallel Haskell libraries from Hackage — follow the instructions in the wiki documentation [DPH].

Moreover, we are working at the third revision of the regular parallel array library [Repa]. It uses indexed types to distinguish multiple array representations, which helps to guide users to write high-performance code. To see it in action, check out Ben Lippmeier's recent demo [Quasicrystals].

Contracts. Work on adding contracts to Haskell, along the lines of Dana Xu's thesis, but using a first order logic theorem prover to check contract satisfaction (with Koen Claessen, Dimitrios Vytiniotis, Charles-Pierre Astolfi, and Nathan Collins).

Liquid types. Ranjit Jhala is working on adding liquid types to GHC. Liquid Types are a form of (dependent) refinement types that use predicate abstraction and SMT solvers to carry out type inference. A prototype has been built that works for a subset of the language (without typeclasses) [Liquid]. Currently, we are working on ways of handling at the basic typeclasses (Ord, Num etc.), and building a web-interface.

Vector instructions. Paul Monday and Geoff Mainland are extending the code generator to exploit vector instructions (with Peter Braam, Duncan Coutts) [VectorInstructions].

A modular package language for Haskell (with Derek Dreyer and Scott Kilpatrick) [Packages].

Bibliography

- [CloudHaskell] Towards Haskell in the Cloud, Jeff Epstein, Andrew P. Black, and Simon Peyton Jones, Haskell Symposium 2011, http://research.microsoft.com/~simonpj/papers/parallel/remote.pdf
- [ConstraintFamilies] Constraint families, Max Bolingbroke, blog post, http://blog.omega-prime.co.uk/?p=61
- [ConstraintKind] Constraint kinds for GHC, Max Bolingbroke, blog post, http://blog.omega-prime.co.uk/?p=127
- [DPH] Data Parallel Haskell documentation, DPH Team, http://haskell.org/haskellwiki/GHC/Data_Parallel_Haskell
- [KindPolymorphism] Giving Hasell a promotion, Brent Yorgey, Stephanie Weirich, Julien Cretin, Dimitrios Vytiniotis, and Simon Peyton Jones, submitted to TLDI'12, http://research.microsoft.com/~simonpj/papers/ext-f/
- [Liquid] Liquid types home page, Ranjit Jhala, http://goto.ucsd.edu/~rjhala/liquid
- [MonadComp] Bringing back monad comprehensions, George Giorgidze, Torsten Grust, Nils Schweinsberg, and Jeroen Weijers, Haskell Symposium 2011, http://db.inf.uni-tuebingen.de/files/giorgidze/haskell2011.pdf
- [NewFC] Practical aspects of evidence-based compilation in System FC, Vytiniotis and Peyton Jones, rejected by ICFP 2011, http://research.microsoft.com/~simonpj/papers/ext-f/
- [Packages] A package language for Haskell, GHC wiki page, http://hackage.haskell.org/trac/ghc/wiki/PackageLanguage
- [ParallelGhcProject] The Parallel GHC Project home page, http://www.haskell.org/haskellwiki/Parallel_GHC_Project
- [Quasicrystals] Quasicrystals Demo, Ben Lippmeier, http://youtu.be/v_0Yyl19fiI
- [Repa] Repa: Regular, shape-polymorphic parallel arrays in Haskell, http://hackage.haskell.org/package/repa
- [SafeHaskell] The Safe Haskell home page, David Terei, http://www.scs.stanford.edu/~davidt/safehaskell.html
- [SHE] The Strathclyde Haskell Enhancement, Conor McBride, 2010, http://personal.cis.strath.ac.uk/~conor/pub/she/
- [ThreadScope] Spark Visualization in ThreadScope, Duncan Coutts, Haskell Implementors workshop 2011, http://www.haskell.org/haskellwiki/HaskellImplementorsWorkshop/2011/Coutts
- [Unleashed] Haskell type constraints unleashed, Tom Schrijvers and Dominic Orchard, FLOPS 2010, http://tomschrijvers.blogspot.com/2009/11/haskell-type-constraints-unleashed.html
- [VectorInstructions] Using SIMD instructions via the LLVM back end, GHC wiki page, http://hackage.haskell.org/trac/ghc/wiki/SimdLlvm
- [WikiConstraint] Adding kind Constraint, GHC wiki page, http://hackage.haskell.org/trac/ghc/wiki/KindFact

3.3 UHC, Utrecht Haskell Compiler

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

What is new? UHC is the Utrecht Haskell Compiler, supporting almost all Haskell98 features and most of Haskell2010, plus experimental extensions. The current focus is on the Javascript backend.

What do we currently do and/or has recently been completed? As part of the UHC project, the following (student) projects and other activities are underway (in arbitrary order):

- Tom Harper (3 month summer stay): Implementing Fusion.
- Jurriën Stutterheim and others: building web applications with the Javascript backend.
- Jeroen Bransen (PhD): "Incremental Global Analysis".
- Jan Rochel (PhD): "Realising Optimal Sharing", based on work by Vincent van Oostrum and Clemens Grabmayer.
- Arie Middelkoop (PhD, defense Jan 2012): type system formalization and automatic generation from type rules, in particular the Attribute Grammar variants Ruler-Core for supporting more complex type system implementations.
- Jeroen Fokker: GRIN backend, whole program analysis.
- Doaitse Swierstra: parser combinator library.
- Atze Dijkstra: overall architecture, type system, bytecode interpreter + java + javascript backend, garbage collector.

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 (→5.4.1).

Further reading

- UHC Homepage: http://www.cs.uu.nl/wiki/UHC/WebHome
- UHC Blog: http://utrechthaskellcompiler.wordpress.com
- Attribute grammar system: http://www.cs.uu.nl/wiki/HUT/AttributeGrammarSystem
- Parser combinators: http://www.cs.uu.nl/wiki/HUT/ParserCombinators
- Shuffle: http://www.cs.uu.nl/wiki/Ehc/Shuffle
- Ruler: http://www.cs.uu.nl/wiki/Ehc/Ruler

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 255 ports of many popular Cabal packages. The updates committed to this repository are continuously integrated to the official ports tree on a regular basis. Though the FreeBSD Ports Collection already has many important Haskell software: GHC 7.0.3, Haskell Platform 2011.2.0.1, Gtk2Hs 0.12, XMonad 0.10, Pandoc 1.8, Darcs 2.5, and Snap 0.5.2 – that is going to be also incorporated into the upcoming FreeBSD 9.0-RELEASE.

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. In total, we maintain 390 source packages, an increase of 80% over the number from the last report.

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 the profiling data and the documentation packages register with the system-wide index.

The transition to GHC 7, which involved renaming all packages, is finished. The stable Debian release ("squeeze") provides the Haskell Platform 2010.1.0.0, Debian testing contains 2011.2.0.1 and in unstable we are currently staging the to-be released 2011.3.0.0. Other noteworthy additions to Haskell on Debian are the yesod packages and a port of GHC to the 64bit mainframe architecture "s390x".

Further reading

http://wiki.debian.org/Haskell

3.4.3 Haskell in Gentoo Linux

Report by:	Sergei Trofimovich

Gentoo Linux currently officially supports GHC 7.0.4 and GHC 6.12.3 on x86, amd64, sparc, alpha and some arms.

The full list of packages available through the official repository can be viewed at http://packages.gentoo.org/category/dev-haskell?full_cat.

The GHC architecture/version matrix is available at http://packages.gentoo.org/package/dev-lang/ghc.

Please report problems in the normal Gentoo bug tracker at bugs.gentoo.org.

There is also an overlay which contains almost 700 extra unofficial and testing packages. Thanks to the Haskell developers using Cabal and Hackage (→ 6.8.1), we have been able to write a tool called "hackport" (initiated by Henning Günther) to generate Gentoo packages with minimal user intervention. Notable packages in the overlay include the latest version of the Haskell Platform (→ 3.1) as well as the latest 7.2.1 release of GHC, as well as popular Haskell packages such as pandoc (→ 8.2.2), gitit (http://www.haskell.org/communities/11-2010/html/report.html#sect5.2.5), yesod (→ 5.2.6) and others.

The team made considerable effort to port a lot of popular packages to ghc-7.2. There is a lot of patches sitting in overlay and waiting for upstream inclusion though.

More information about the Gentoo Haskell Overlay can be found at http://haskell.org/haskellwiki/Gentoo. It is available via the Gentoo overlay manager "layman". If you choose to use the overlay, then any problems should be reported on IRC (`#gentoo-haskell` on freenode), where we coordinate development, or via email ⟨haskell@gentoo.org⟩ (as we have more people with the ability to fix the overlay packages that are contactable in the IRC channel than via the bug tracker).

As always we are more than happy for (and in fact encourage) Gentoo users to get involved and help us maintain our tools and packages, even if it is as simple as reporting packages that do not always work or need updating: with such a wide range of GHC and package versions to co-ordinate, it is hard to keep up! Please contact us on IRC or email if you are interested!

For concrete tasks see our perpetual TODO list: https://github.com/gentoo-haskell/gentoo-haskell/blob/master/projects/doc/TODO.rst

3.4.4 Fedora Haskell SIG

Report by:	Jens Petersen
Participants:	Lakshmi Narasimhan, Ben Boeckel, Michel Salim, Shakthi Kannan, Bryan O'Sullivan, and others
Status:	ongoing

The Fedora Haskell SIG is an effort to provide good support for Haskell in Fedora.

Fedora 16 is shipping early in November with ghc-7.0.4 and haskell-platform-2011.2.0.1, and updates to many of the packages. Newly added packages this time include leksah, cabal-dev, cab, and over 25 new libraries.

There are some packaging improvements:

- All Haskell interdependencies are now purely automatically generated, with ABI hashes.
- Profiling subpackages for libraries have been merged into the library development subpackages to simplify packaging, installation, and maintenance: so there are no prof subpackages in Fedora 16.

These changes have also been partially backported to Fedora 14 and 15.

Fedora's Haskell packages have been ported to some new architectures:

- I forgot to mention in the last report that Fabio M. Di Nitto had ported most of Fedora 15 Haskell Platform to sparcv9.
- Jiri Skala has ported most of Fedora 15's Haskell Platform to ppc64, in addition to the existing ppc port.
- I am also excited that Henrik NordstrÃűm has been working on porting Fedora 15's ghc-7.0.2 to armv7hl and armv5tel.

From these ports have also followed various ghc bootstrapping packaging improvements.

There are currently 139 Haskell source packages in Fedora. Note the Fedora package version numbers listed on the Hackage website refer to the packages for the latest stable Fedora release.

In the Fedora 17 development cycle it is planned to update ghc to 7.4 and continue work on packaging the Snap and Yesod web frameworks.

Feedback from users and packaging contributions to Fedora Haskell are always welcome: join us on #fedora-haskell on Freenode IRC and our mailing-list.

Further reading

- Homepage: http://fedoraproject.org/wiki/SIGs/Haskell

- Fedora 16 Haskell release-notes: http://fedoraproject.org/wiki/Documentation_Development_Haskell_Beat
- Package list: https://admin.fedoraproject.org/pkgdb/users/packages/haskell-sig?tg_paginate_limit=0
- Dependency graphs: https://fedoraproject.org/wiki/Haskell_package_interdependencies

3.5 Fibon Benchmark Tools & Suite

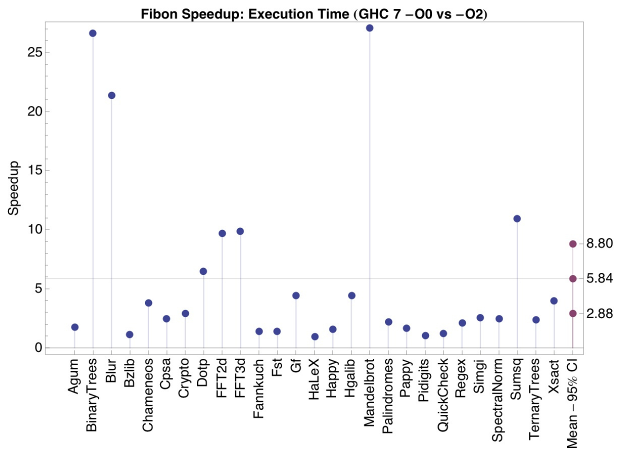
Report by:	David M. Peixotto
Status:	stable

Fibon is a set of tools for running and analyzing benchmark programs in Haskell. It contains an optional set of benchmarks from various sources including several programs from the Hackage repository.

The Fibon benchmark tools draw inspiration from both the venerable nofib Haskell benchmark suite and the industry standard SPEC benchmark suite. The tools automate the tedious parts of benchmarking: building the benchmark in a sand-boxed directory, running the benchmark multiple times, verifying correctness, collecting statistics, and summarizing results.

Benchmarks are built using the standard `cabal` tool. Any program that has been cabalized can be added as benchmark simply by specifying some meta-information about the program inputs and expected outputs. Fibon will automatically collect execution times for benchmarks and can optionally read the statistics output by the GHC runtime. The program outputs are checked to ensure correct results making Fibon a good option for testing the safety and performance of program optimizations. The Fibon tools are not tied to any one benchmark suite. As long as the correct meta-information has been supplied, the tools will work with any set of programs.

As a real life example of a complete benchmark suite, Fibon comes with its own set of benchmarks for testing the effectiveness of compiler optimizations in GHC. The benchmark programs come from Hackage, the Computer Language Shootout, Data Parallel Haskell, and Repa. The benchmarks were selected to have minimal external dependencies so they could be easily used with a version of GHC compiled from the latest sources. The following figure shows the performance improvement of GHC's optimizations on the Fibon benchmark suite.



This year, the Fibon benchmark suite has been updated to include a `Train` problem size that can be used for feedback directed optimization work. The `Ref` problem size has been increased so that the running time of a benchmark program is comparable to the running time when using the ref size of the SPEC benchmarks. With this update a single benchmark will typically take $10 - 30$ minutes to run depending on the power of the computer hardware. See the README file for more information on benchmark size and configuring the benchmarks to finish in an acceptable amount of time.

The Fibon tools and benchmark suite are ready for public consumption. They can be found on github at the url indicated below. People are invited to use the included benchmark suite or just use the tools and build a suite of their own creation. Any improvements to the tools or additional benchmarks are most welcome. Benchmarks have been used to tell lies about performance for many years, so join in the fun and keep on fibbing with Fibon.

Further reading

- https://github.com/dmpots/fibon
- https://github.com/dmpots/fibon-benchmarks
- https://github.com/dmpots/fibon-config

4 Related Languages

4.1 Agda

Report by:	Nils Anders Danielsson
Participants:	Ulf Norell, Andreas Abel, and many others
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.

At the time of writing version 2.3.0 is about to be released, with the following new features (among others):

- Instance arguments (Dominique Devriese).
- A JavaScript backend (Alan Jeffrey).
- More optimizations in the Epic backend (Olle Fredriksson and Daniel Gustafsson).
- Pattern matching, multi-clause lambdas (Fredrik Nordvall Forsberg, Karim Kanso and Noam Zeilberger).

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. 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.

Recent features include bounded size quantification and destructor patterns for a more general handling of coinduction. In the long run, I plan to evolve MiniAgda into a core language for Agda with termination certificates.

MiniAgda is available as Haskell source code and compiles with GHC $\geqslant$ 6.12.x.

Further reading

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

4.3 Clean

Report by:	Thomas van Noort
Participants:	Rinus Plasmeijer, John van Groningen
Status:	active development

Clean is a general purpose, state-of-the-art, pure and lazy functional programming language designed for making real-world applications. Here is a short list of notable features:

- Clean is a lazy, pure, and higher-order functional programming language with explicit graph-rewriting semantics.
- Although Clean is by default a lazy language, one can smoothly turn it into a strict language to obtain optimal time/space behavior: functions can be defined lazy as well as (partially) strict in their arguments; any (recursive) data structure can be defined lazy as well as (partially) strict in any of its arguments.
- Clean is a strongly typed language based on an extension of the well-known Milner/Hindley/Mycroft type inferencing/checking scheme including the common higher-order types, polymorphic types, abstract types, algebraic types, type synonyms, and existentially quantified types.
- Clean has pattern matching, guards, list comprehensions, array comprehensions and a lay-out sensitive mode.
- Clean supports type classes and type constructor classes to make overloaded use of functions and operators possible.
- The uniqueness typing system in Clean makes it possible to develop efficient applications. In particular, it allows a refined control over the single-threaded use of objects which can influence the time and space behavior of programs. Uniqueness typing can also be used to incorporate destructive updates of objects

within a pure functional framework. It allows destructive transformation of state information and enables efficient interfacing to the nonfunctional world (to C but also to I/O systems like X-Windows) offering direct access to file systems and operating systems.

- Clean offers records and (destructively updateable) arrays and files.
- The Clean type system supports dynamic typing, allowing values of arbitrary types to be wrapped in a uniform package and unwrapped via a type annotation at run time. Using dynamics, code and data can be exchanged between Clean applications in a flexible and type-safe way.
- Clean provides a built-in mechanism for generic functions.
- There is a Clean IDE and there are many libraries available offering additional functionality.
- There is (experimental) support for the exchange of sources between Clean and Haskell, please see http://www.haskell.org/communities/11-2010/html/report.html#sect3.6 for more information and future plans.

Future plans

- We are currently working on the generic function mechanism: we are improving efficiency and including support for generic dependencies, the latter allows us to use arbitrary generic functions on the type parameters of a generic type argument.
- Clean is already available for 32-bit and 64-bit Windows and Linux, we are currently working on 64-bit Mac support.

Further reading

- http://wiki.clean.cs.ru.nl/
- http://wiki.clean.cs.ru.nl/Download_Clean

4.4 Timber

Report by:	Johan Nordlander
Participants:	Björn von Sydow, Andy Gill, Magnus Carlsson, Per Lindgren, Thomas Hallgren, and others
Status:	actively developed

See: http://www.haskell.org/communities/11-2010/html/report.html#sect4.5.

4.5 Disciple

Report by:	Ben Lippmeier
Participants:	Erik de Castro Lopo
Status:	experimental, active development

Disciple is a dialect of Haskell that uses strict evaluation as the default and supports destructive update of arbitrary data. Many Haskell programs are also Disciple programs, or will run with minor changes. In addition, Disciple includes region, effect, and closure typing, and this extra information provides a handle on the operational behaviour of code that is not available in other languages. Our target applications are the ones that you always find yourself writing C programs for, because existing functional languages are too slow, use too much memory, or do not let you update the data that you need to.

Our compiler (DDC) is still in the "research prototype" stage, meaning that it will compile programs if you are nice to it, but expect compiler panics and missing features. You will get panics due to ungraceful handling of errors in the source code, but valid programs should compile ok. The test suite includes a few thousand-line graphical demos, like a ray-tracer and an n-body collision simulation, so it is definitely hackable.

Over the last six months we continued working towards mechanising the metatheory of the DDC core language in Coq. We've finished Progress and Preservation for System-F2 with mutable algebraic data, and are now looking into proving contextual equivalence of rewrites in the presence of effects. Based on this experience, we've also started on an interpreter for a cleaned up version of the DDC core language. We've taken the advice of previous paper reviewers and removed dependent kinds, moving witness expressions down to level 0 next to value expressions. In the resulting language, types classify both witness and value expressions, and kinds classify types. We're also removing more-than constraints on effect and closure variables, along with dangerous type variables (which never really worked). All over, it's being pruned back to the parts we understand properly, and the removal of dependent kinds will make mechanising the metatheory easier. Writing an interpreter for the core language also gets us a parser for it, which we will need for performing cross module inlining in the compiler proper.

Further reading

http://disciple.ouroborus.net

5 Haskell and . . .

5.1 Haskell and Parallelism

5.1.1 Eden

Report by:	Rita Loogen
Participants:	**in Madrid:** Yolanda Ortega-Mallén, Mercedes Hidalgo, Lidia Sánchez-Gil, Fernando Rubio, Alberto de la Encina, **in Marburg:** Mischa Dieterle, Thomas Horstmeyer, Oleg Lobachev, Rita Loogen, Bernhard Pickenbrock **in Copenhagen:** 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 main constructs are process abstractions and process instantiations. The Eden logo

consists of four λ turned in such a way that they form the Eden instantiation operator #05. 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, Draft Lecture Notes, CEFP Summer School, Budapest, Hungary, June 2011.
(see also: http://www.mathematik.uni-marburg.de/~eden/?content=cefp)

Implementation

The current release of the Eden compiler based on GHC 6.12.3 is available on our web pages, see http://www.mathematik.uni-marburg.de/~eden. A release based on GHC 7 is in preparation. It will include a shared memory mode which does not depend on a middleware like MPI but which nevertheless uses multiple independent heaps (in contrast to GHC's threaded runtime system) connected by Eden's parallel runtime system. An Eden variant of GHC's head version is available in a repository on github, see https://github.com/jberthold/ghc.

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 has been written in Haskell and is freely available on the Eden web pages.

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

- Oleg Lobachev: *Implementation and Evaluation of Algorithmic Skeletons: Parallelisation of Computer Algebra Algorithms*, Ph.D. thesis, Philipps-Universität Marburg, Germany, October 2011.
- Rita Loogen: *Eden - Parallel Functional Programming in Haskell*, Draft Lecture Notes, CEFP Summer School, Budapest, Hungary, June 2011.
- B. Pickenbrock: *A Multicore Implementation of Eden*, Bachelor thesis, Philipps-Universität Marburg, 2011 (in German).
- Jost Berthold, Andrzej Filinski, Fritz Henglein, Ken Friis Larsen, Mogens Steffensen, and Brian Vinter: *Functional High Performance Financial IT — The HIPERFIT Research Center in Copenhagen*, Trends in Functional Programming (TFP'11) — Revised Selected Papers, Springer LNCS (to appear).
- J. Berthold: *Orthogonal Serialisation for Haskell*, 22nd Symposium on Implementation and Application of Functional Languages (IFL 2010), Springer LNCS 6647, pages 38-53, 2011.
- C. Brown, H.-W. Loidl, J. Berthold, and K. Hammond: *Improve your CASH flow: The Computer Algebra SHell*, In 22nd Symposium on Implementation and Application of Functional Languages (IFL 2010), Springer LNCS 6647, pages 169-184, 2011.

Further reading

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

5.1.2 GpH — Glasgow Parallel Haskell

Report by:	Hans-Wolfgang Loidl
Participants:	Phil Trinder, Patrick Maier, Mustafa Aswad, Malak Aljabri, Robert Stewart (Heriot-Watt University); Kevin Hammond, Vladimir Janjic, Chris Brown (St Andrews University)
Status:	ongoing

Status

A distributed-memory, GHC-based implementation of the parallel Haskell extension GpH and of a fundamentally revised version of the evaluation strategies abstraction is available in a prototype version. In current research an extended set of primitives, supporting hierarchical architectures of parallel machines, and extensions of the runtime-system for supporting these architectures are being developed.

System Evaluation and Enhancement

- Both GpH and Eden (→ 5.1.1) parallel Haskells are being used for parallel language research and in the SCIEnce and HPC-GAP projects (see below).
- We are extending the set of primitives for parallelism to better control data locality.
- We are revising the evaluation strategies abstraction for improved genericity.
- We are teaching parallelism to undergraduates using GpH at Heriot-Watt and Phillips Universität Marburg.

GpH Applications

As part of the SCIEnce EU FP6 I3 project (026133) (April 2006 – December 2011) and the HPC-GAP project (October 2009 – September 2013) we use Eden and GpH as middleware to provide access to computational Grids from Computer Algebra (CA) systems, including GAP, Maple MuPad and KANT. We have developed and released SymGrid-Par, a Haskell-side infrastructure for orchestrating heterogeneous computations across high-performance computational Grids. Based on this infrastructure we have developed a range of domain-specific parallel skeletons for parallelising representative symbolic computation applications. We are currently extending SymGrid-Par with support for fault-tolerance, targeting massively parallel high-performance architectures.

In recent work we have developed and released a GHCi-based computer algebra shell (CASH) that gives direct access to computer algebra functionality, provided by an SCSCP server, and enabling easy parallelism on the Haskell side.

Implementations

The latest GUM implementation of GpH is built on GHC 6.12, using either PVM or MPI as communications library. It implements a virtual shared memory abstraction over a collection of physically distributed machines. At the moment our main hardware platforms are Intel-based Beowulf clusters of multicores. We plan to connect several of these clusters into a wide-area, hierarchical, heterogenous parallel architecture.

Further reading

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

Contact

⟨gph@macs.hw.ac.uk⟩

5.1.3 Parallel GHC project

Report by:	Eric Kow
Participants:	Duncan Coutts, Andres Löh, Nicolas Wu, Mikolaj Konarski
Status:	active

Microsoft Research is funding a 2-year project to promote the real-world use of parallel Haskell. The project started in November 2010, with four industrial partners, and consulting and engineering support from Well-Typed (→ 9.1). Each organisation is working on its own particular project making use of parallel Haskell. The overall goal is 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.

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
- Willow Garage Inc. (USA): Distributed Rigid Body dynamics in ROS (Robot Operating System) on clusters
- IIJ Innovation Institute Inc. (Japan): network servers handling a massive number of concurrent connections

Since the last report, the Parallel GHC project has been joined by two new industrial partners, the research and development group of Spanish telecoms company Telefonica, and VETT a UK-based payment processing company. We are excited to be working with the teams at Telefonica I+D and VETT. We hope to making good use of Cloud Haskell with these partners.

Meanwhile, there has been a lot of work in documenting and presenting the work done in the project to the world. The team at Los Alamos National Laboratory have presented to their colleagues their work on high performance Monte Carlo simulations using parallel Haskell, now published under in the report LA-UR 11-0341. Duncan Coutts presented our recent work on ThreadScope at the Haskell Implementors Workshop in Tokyo (23 Sep). He talked about the new spark visualisation feature which shows a graphical representation of spark creation and conversion statistics.

The project has also inspired some interesting write-ups from project members working on the side. Bernie Pope and Dmitry Astapov wrote an article for the recent Monad Reader special edition on parallelism and concurrency. In their article, Bernie and Dmitry discuss the Haskell MPI binding developed within the context of this project. Kazu Yamamoto wrote an article discusses the latest version of the high-performance web server Mighttpd, particularly how it takes advantage of the new IO manager in GHC 7.

In addition to documentation, the project has also made a few software releases

- ThreadScope (0.2.0), with a new spark profiling visualisation, bookmarks, and other enhancements
- Gtk2Hs (0.12.1), adding GHC 7.2 compatibility,
- ghc-events (0.3.0.1), adding support for spark events (GHC 7.3 needed for this)

Finally, we have completed a pure Haskell implementation of the "Modified Additive Lagged Fibonacci" random number generator. This generator is attractive for use in Monte Carlo simulations because it is splittable and has good statistical quality, while providing high performance. The LFG implementation will be released on Hackage when it has undergone more extensive quality testing.

5.2 Haskell and the Web

5.2.1 WAI

Report by:	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 is also a platform for sharing 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. WAI applications can send a response themselves. For example, wai-app-static is used by Yesod to serve static files. By targeting WAI, every web framework can share WAI code instead of wasting effort re-implementing the same functionality.

WAI is most often used in conjunction with the Yesod web framework (→ 5.2.6), but it is designed in a framework independent way. There are some plain WAI users such as Hoogle (→ 6.2.2). There are also some new web frameworks that take a completely different approach to web development that use WAI, such as webwire (FRP) and dingo (GUI).

The WAI standard has proven itself capable for different users and there are no major plans for changes and improvements. Future ideas include allowing Middleware to pass along arbitrary data.

Further reading

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

5.2.2 Warp

Report by:	Greg Weber

Warp is a high performance, easy to deploy HTTP server backend for WAI (→ 5.2.1). Since the last HCAR, Warp has become more battle tested and can be considered a stable, production ready web server. Due to the combined use of ByteStrings, Blaze-Builder, Enumerators, and GHC's improved I/O manager, Wai+Warp has consistently proven to be Haskell's most performant web deployment option. Its performance is better than dynamic language alternatives and seems to be in league with industry standards such as Nginx (benchmarks forthcoming). Warp currently serves Hoogle (→ 6.2.2), hums, and several production Yesod web sites (→ 5.2.6).

"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

5.2.3 Holumbus Search Engine Framework

Report by:	Uwe Schmidt
Participants:	Timo B. Hübel, Sebastian Gauck, Stefan Schmidt, Sebastian Schröder
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

There are two running projects. The first, a masters thesis done by Sebastian Schröder, deals with the development of a framework for news systems. The functionality will be like with google news, but the target is to build news systems for specialized topics. We expect to finish this project at the end of 2011.

In the second project a specialized search engine for our university web site has been built. The new aspect in this application is a specialized free text search for appointments, deadlines, announcements, meetings and other dates. There is a running prototype of this search engine. We expect to finish this work in November 2011 and then to use this engine as the official search engine of our university web site.

Further reading

The Holumbus web page (http://holumbus.fh-wedel.de/) includes downloads, Git web interface, current status, requirements, and documentation. Timo Hübel'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

The Happstack project is focused on leveraging the unique characteristics of Haskell to create a highly-scalable, robust, and expressive web framework.

While Happstack is over 7 years old, it is still undergoing active development and new innovation. It is used in a number of commercial projects as well as the new Hackage 2 server.

At the core of Happstack is the happstack-server package which provides a fast, powerful, and easy to use HTTP server with built-in support for templating (via blaze-html), request routing, form-decoding, cookies, file-uploads, etc. happstack-server is all you need to create a simple website.

Happstack can also be extended using a wide range of libraries which include support for alternative HTML templating systems, javascript templating and generation, type-safe URLs, type-safe form generation and validation, RAM-cloud database persistence, OpenId authentication, and more.

Future plans

Upcoming innovations we will be exploring in Happstack include:

- more powerful and flexible 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
- better support for reusable web components (such as components for authentication, threaded comments, etc)
- fundamental architecture changes to the backend which will allow for greater scalability

We will be blogging about our findings and soliciting feedback.

Further reading

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

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 providing basic web features and CGI (mailman and contents search).

Mighttpd version 1 was implemented with two libraries *c10k* and *webserver*. Since GHC 6 uses select(), more than 1,024 connections cannot be handled at the same time. The *c10k* library gets over this barrier with the pre-fork technique. The *webserver* library provides HTTP transfer and file/CGI handling.

Mighttpd 2 stops using the *c10k* library because GHC 7 starts using epoll()/kqueue(). The file/CGI handling part of the *webserver* library is reimplemented as a web application on the *wai* library (→ 5.2.1). For HTTP transfer, Mighttpd 2 links the *warp* library (→ 5.2.2) which can send a file in zero copy manner thank to sendfile().

The performance of Mighttpd 2 is now comparable to highly tuned web servers written in C Please read “The Monad.Reader” Issue 19 for more information.

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

Further reading

http://www.mew.org/~kazu/proj/mighttpd/en/

5.2.6 Yesod

Report by:	Greg Weber
Participants:	Michael Snoyman, Luite Stegeman, Patrick Brisbin
Status:	stable

Yesod is a web framework that helps users create highly scalable web applications.

Performance scalablity comes from the amazing GHC compiler and runtime. GHC provides fast code and built-in evented asynchronous IO. The standard Warp web server utilizes this to serve more simlutaneous requests than any other web application server we know of.

But Yesod is even more focused on scalable development. A developer should be able to continue to productively write code as their application grows and more team members join, including designers. 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.

- type-safe urls - say goodbye to broken links
- no XSS attacks - user submissions are automatically sanitized
- no SQL injection - sql queryies are automatically escaped
- database queries are always valid - querying is done in Haskell and uses your schema
- valid template variables with proper template insertion- variable are known at compile time and treated differently according to their type

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 (→ 7.4.2). Views are handled by the Shakespeare family of compile-time template languages. This includes Hamlet, which takes the tedium out of HTML. Controllers are invoked through declarative routing. Their return type shows which response types are allowed for the request.

Yesod is broken up into many smaller projects and uses (→ 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. Recently a continuation-based FRP web framework called webwire was released. It uses WAI and many other libraries that have been produced under Yesod.

Yesod is currently on its 0.9 version. The last HCAR entry was for the 0.8 version. Since then we have added:

- A complete i18n (internationalization) solution.
- A drasticaly improved development environment that automatically recompiles and restarts your application

◦ An overhaul of the form system for more flexibility and better error messages

◦ An overhaul of the template system for better error messages

◦ The addition of shakespeare-text for easy string combination

◦ An integrated logging system along with great development request logging.

◦ GHC7.2 support while maintaining GHC6 compatibility

◦ support for coffeescript (a better javascript) templates.

◦ Static file serving with automatic caching headers

◦ Support for asynchronous javascript loading.

◦ Runtime configuration settings loaded from YAML files.

◦ Posix signal handling

We are excited to be near a 1.0 release. 1.0 to us means API stability and a web framework that gives developers all the tools they need for productive web development. But we already have a productive framework in use by the Haskell community, including commercial users.

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

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.

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, 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.

The Snap Framework has seen two major releases (0.5 and 0.6) since the last HCAR with a development team that continues to grow. Snap 0.6 introduces composable web application components called snaplets, which allow you to build self-contained pieces of your web site in a structured way. The snaplet API simplifies distribution, installation, and configuration, allowing you to easily add new functionality to your application in a safe, clean way with very little boilerplate. Snap 0.6 also ships with built-in snaplets for templating, sessions, and authentication.

In September, Gregory Collins gave a CUFP tutorial on building web applications with Snap. The tutorial demonstrated how to use long polling JSON calls to implement a simple web-based chat room. Slides and source code for his presentation are in the links below.

Since the 0.6 release an independently written snaplet for accessing HDBC databases has already been published. We expect to see more of this kind of third-party development and hope to eventually have a vibrant ecosystem of snaplets providing a deep body of pluggable functionality.

Further reading

◦ Snaplet Tutorial: http://snapframework.com/docs/tutorials/snaplets-tutorial
◦ Snaplet Directory: http://snapframework.com/snaplets
◦ CUFP Tutorial Slides: http://gregorycollins.net/posts/2011/10/01/cufp-tutorial-slides
◦ CUFP Tutorial Source Code: https://github.com/snapframework/cufp2011
◦ http://snapframework.com

5.2.8 Ivy-web

Report by:	James Deng
Status:	experimental

Ivy-web is a lightweight web framework, with type safe routes, based on invertible-syntax, and i18n support, influenced by Django, Snap, and Yesod.

The features of this web framework:

◦ Type safe routes, specify url-handler mapping in one place. For example, we want a url mapping for blog as "/blog/year-month-day" to Handler Int Int Int, where year, month and day are integers. We can declare as follows:

```
data Blog = Blog Int Int Int
              deriving (Show, Eq, Typeable)
$ (defineIsomorphisms '' Blog)
instance Handler Blog where
  get b@(Blog y m d) _ = do
    t ← liftIO getClockTime
    return $ responseHtml $ trans′ "blog"
             ++ show b ++ show t
```

$$rBlog = blog <\$> text \texttt{"/blog/"} *> int <-> int <-> int$$

We can reverse this mapping from handler value automatically, thus do not need to construct url string manually in code, avoiding url errors.

```
ghci>  url (Blog 2011 9 19)
       == "/blog/2009-9-19"
```

- Simple yet elegant handler via type class.

```
class Handler a where
  get, post, put, delete, handle :: a → Application
  handle a req = case requestMethod req of
    m | m ≡ methodGet → get a req
      | m ≡ methodPost → post a req
      | m ≡ methodPut → put a req
      | m ≡ methodDelete → delete a req
    otherwise → unimplemented req
```

- Flexible template system, utilize exsisting libraries such as Blaze-Html and Hastache.
- Easy i18n — Wraps around i18n library.
- TODO: Auth system — Port from snap-auth.
- TODO: Modular app system like Django — The current route system support modular routes very well. Need works in modular config and data files like static template files.
- TODO: Persistent library — Improving the DSH library is my current preference.

The principle of this library is KISS, and "don't reinvent the wheel" by reusing existing state-of-the-art libraries.

For the example code listed above, please refer to https://github.com/lilac/ivy-example/

Recent developments

I have ported ivy-web from wai to snap-server backend, and also wrote a sample project correspond to the starter project of snap. When everything is fine and I am free, I will upload the code and bump the version to 0.2.

Further reading

- https://github.com/lilac/ivy-web/
- http://hackage.haskell.org/package/ivy-web

5.2.9 rss2irc

Report by:	Simon Michael
Status:	beta

rss2irc is an IRC bot that polls a single RSS or Atom feed and announces new items to an IRC channel, with options for customizing output and behavior. It aims to be an easy to use, dependable bot that does its job and creates no problems.

rss2irc was published in 2008 by Don Stewart. Simon Michael took over maintainership in 2009, with the goal of making a robust low-maintenance bot to stimulate development in various free/open-source software communities. It is currently used for several full-time bots including:

- hackagebot — announces new hackage releases in #haskell
- hledgerbot — announces hledger commits in #ledger
- zwikicommitbot — announces Zwiki commits in #zwiki
- squeaksobot — announces Squeak and Smalltalk-related Stack Overflow questions in #squeak
- squeakquorabot — announces Squeak/Smalltalk-related Quora questions in #squeak
- etoystrackerbot — announces new Etoys bugs in #etoys
- etoysupdatesbot — announces Etoys commits in #etoys
- planetzopebot — announces new planet.zope.org posts in #zope

The project is available under BSD license from its home page at http://hackage.haskell.org/package/rss2irc.

Since last report there has been a great deal of cleanup and enhancement, but no new release on hackage yet due to an xml-related memory leak.

Further reading

http://hackage.haskell.org/package/rss2irc

5.3 Haskell and Games

5.3.1 FunGEn

Report by:	Simon Michael
Status:	usable; ready for contributors and users

FunGEn (Functional Game Engine) is a BSD-licensed cross-platform 2D game engine implemented in and for Haskell, using OpenGL and GLUT. It was created in 2002 by Andre Furtado, updated in 2008 by Simon Michael and Miloslav Raus, and revived again in 2011, with a GHC 6.12-tested 0.3 release on Hackage, preliminary haddockification and a new home repo.

FunGEn remains the quickest path to building cross-platform graphical games in Haskell, due to its convenient game framework and widely-available dependencies. It comes with several working examples that are quite easy to read and build (pong, worms). In the last six months there has been little activity and a new maintainer would be welcome.

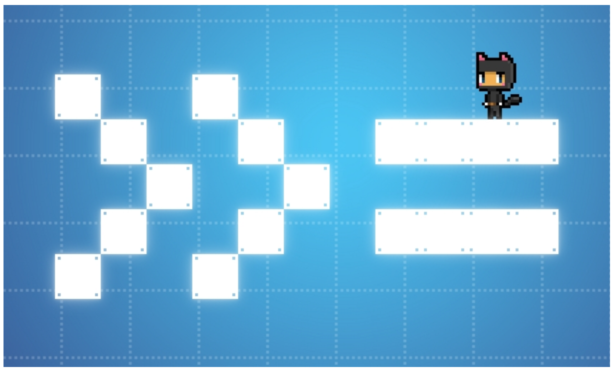
FunGEn-related discussions most often appear in the #haskell-game channel on irc.freenode.net.

Further reading

http://darcsden.com/simon/fungen

5.3.2 Nikki and the Robots

Report by:	Sönke Hahn
Participants:	Joyride Laboratories GbR
Status:	alpha, active



Nikki and the Robots is a 2D platformer written in Haskell and produced by Joyride Laboratories. Nikki, the protagonist, walks and jumps around the levels wearing a cute ninja/cat costume. Nikki refrains from using any tools or weapons, with one exception: The Robots. These come in various types with different abilities and can be used by Nikki to solve puzzles, overcome obstacles, and complete the level tasks. The game features an integrated level editor.

We made our first binary release of Nikki and the Robots in April 2011.

Publishing

We are releasing the game and the level editor under an open source license (LGPL). The included graphics are published under a permissive Creative Commons license (cc-by-sa). We are also planning to create a server that will allow players to upload the levels they created and download levels from other players. We hope that a community of coders, level creators, and players will emerge around the game.

Simultaneously, we are working on episodes that we plan to sell via the game. These will include new graphics, more robots, a story line, other characters, and other surprises.

(Just to clarify: The licensing is very permissive. It allows others to create their own episodes and distribute them freely or sell them. This would be very welcome. If anybody is interested in this, we propose to join forces and sell all our episodes through one system.)

Technologies Used

- Qt for user input and rendering.
- OpenGL as an efficient rendering backend for Qt. Everything will remain 2D, though - we promise!
- Hipmunk, the Haskell bindings to the chipmunk physics engine.

Getting Involved

The project is still in alpha stage, so there are some features that are not yet implemented. For some, we have a clear vision on how to implement them; for others, we do not. If you want to get involved, check out our darcs repo, our launchpad site, and do not hesitate to contact us.

Further reading

- http://joyridelabs.de
- http://joyridelabs.de/game/code/
- http://joyridelabs.de/game/download/

5.4 Haskell and Compiler Writing

5.4.1 UUAG

Report by:	Arie Middelkoop
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 (→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 0.9.39 (October 2011), is extensively tested, and is available on Hackage. Recently, we improved the Cabal support and ensured compatibility with GHC 7.

We are working on the following enhancements of the UUAG system:

First-class AGs We provide a translation from UUAG to AspectAG (→5.4.2). AspectAG is a library of strongly typed Attribute Grammars implemented using type-level programming. With this extension, we can write the main part of an AG conveniently with UUAG, and use AspectAG for (dynamic) extensions. Our goal is to have an extensible version of the UHC.

Ordered evaluation We have implemented a variant of Kennedy and Warren (1976) for *ordered* AGs. For any absolutely non-circular AGs, this algorithm finds a static evaluation order, which solves some of the problems we had with an earlier approach for ordered AGs. A static evaluation order allows the generated code to be strict, which is important to reduce the memory usage when dealing with large ASTs. The generated code is purely functional, does not require type annotations for local attributes, and the Haskell compiler proves that the static evaluation order is correct.

Multi-core evaluation Our algorithm for ordered AGs identifies statically which subcomputations of children of a production are independent and suitable for parallel evaluation. Together with the strict evaluation as mentioned above, which is important when evaluating in parallel, the generated code can automatically exploit multi-core CPUs. We are currently evaluating the effectiveness of this approach.

Stepwise evaluation In the recent past we worked on a stepwise evaluation scheme for AGs. Using this scheme, the evaluation of a node may yield user-defined progress reports, and the evaluation to the next report is considered to be an evaluation step. By asking nodes to yield reports, we can encode the parallel exploration of trees and encode breadth-first search strategies.

We are currently also running a Ph.D. project that investigates incremental evaluation.

Further reading

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

5.4.2 AspectAG

Report by:	Marcos Viera
Participants:	Doaitse Swierstra, Wouter Swierstra
Status:	experimental

AspectAG is a library of strongly typed Attribute Grammars implemented using type-level programming.

Introduction

Attribute Grammars (AGs), a general-purpose formalism for describing recursive computations over data types, avoid the trade-off which arises when building software incrementally: should it be easy to add new data types and data type alternatives or to add new operations on existing data types? However, AGs are usually implemented as a pre-processor, leaving e.g. type checking to later processing phases and making interactive development, proper error reporting and debugging difficult. Embedding AG into Haskell as a combinator library solves these problems. Previous attempts at embedding AGs as a domain-specific language were based on extensible records and thus exploiting Haskell's type system to check the well-formedness of the AG, but fell short in compactness and the possibility to abstract over oft occurring AG patterns. Other attempts used a very generic mapping for which the AG well-formedness could not be statically checked. We present a typed embedding of AG in Haskell satisfying all these requirements. The key lies in using HList-like typed heterogeneous collections (extensible polymorphic records) and expressing AG well-formedness conditions as type-level predicates (i.e., typeclass constraints). By further type-level programming we can also express common programming patterns, corresponding to the typical use cases of monads such as Reader, Writer, and State. The paper presents a realistic example of type-class-based type-level programming in Haskell.

We have included support for local and higher-order attributes. Furthermore, a translation from UUAG to AspectAG is added to UUAGC as an experimental feature.

Current Status

We have recently added a combinator *agMacro* to provide support for "attribute grammars macros"; a mechanism that makes it easy to define attribute computation in terms of already existing attribute computation.

Background

The approach taken in AspectAG was proposed by Marcos Viera, Doaitse Swierstra, and Wouter Swierstra in the ICFP 2009 paper "Attribute Grammars Fly First-Class: How to do aspect oriented programming in Haskell".

The Attribute Grammar Macros combinator is described in a technical report: UU-CS-2011-028.

Further reading

http://www.cs.uu.nl/wiki/bin/view/Center/AspectAG

5.4.3 LQPL — A Quantum Programming Language Compiler and Emulator

Report by:	Brett G. Giles
Participants:	Dr. J.R.B. Cockett
Status:	v 0.8.4 experimental released

See: http://www.haskell.org/communities/11-2010/html/report.html#sect5.4.4.

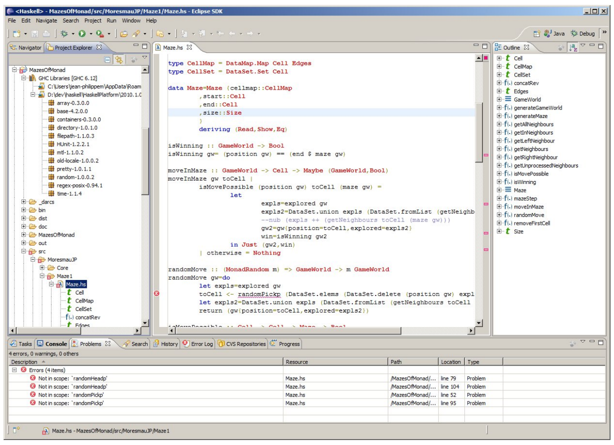
6 Development Tools

6.1 Environments

6.1.1 EclipseFP

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

EclipseFP is a set of Eclipse plugins to allow working on Haskell code projects. It features Cabal integration (.cabal file editor, uses Cabal settings for compilation), and GHC integration. Compilation is done via the GHC API, syntax coloring uses the GHC Lexer. Other standard Eclipse features like code outline, folding, and quick fixes for common errors are also provided. EclipseFP also allows launching GHCi sessions on any module including extensive debugging facilities. It uses Scion to bridge between the Java code for Eclipse and the Haskell APIs. The source code is fully open source (Eclipse License) and anyone can contribute. Current version is 2.1.0, released in September 2011 and supporting GHC 6.12 and 7.0, and more versions with additional features are planned. 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.



```
type CellMap = DataMap.Map Cell Edges
type CellSet = DataSet.Set Cell

data Maze=Maze {cellmap::CellMap
        ,start::Cell
        ,end::Cell
        ,size::Size
        }
        deriving (Read,Show,Eq)

isWinning :: GameWorld -> Bool
isWinning gw= (position gw) == (end $ maze gw)

moveInMaze :: GameWorld -> Cell -> Maybe (GameWorld,Bool)
moveInMaze gw toCell |
        isMovePossible (position gw) toCell (maze gw) =
                let
                        expls=explored gw
                        expls2=DataSet.union expls (DataSet.fromList (getNeighb
                        --nub (expls ++ (getNeighbours toCell (maze gw)))
                        gw2=gw{position=toCell,explored=expls2}
                        win=isWinning gw2
                in Just (gw2,win)
        | otherwise = Nothing

randomMove :: (MonadRandom m) => GameWorld -> m GameWorld
randomMove gw=do
        let expls=explored gw
        toCell <- randomPickp (DataSet.elems (DataSet.delete (position gw) expl
        let expls2=DataSet.union expls (DataSet.fromList (getNeighbours toCell
        return (gw{position=toCell,explored=expls2})
```

4 errors, 0 warnings, 0 others

Description	Resource	Path	Location	Type
Errors (4 items)				
Not in scope: `randomHeadp'	Maze.hs	/MazesOfMonad/...	line 79	Problem
Not in scope: `randomHeadp'	Maze.hs	/MazesOfMonad/...	line 104	Problem
Not in scope: `randomPickp'	Maze.hs	/MazesOfMonad/...	line 52	Problem
Not in scope: `randomPickp'	Maze.hs	/MazesOfMonad/...	line 95	Problem

Maze.hs - MazesOfMonad/src/MoresmauJP/Maze1

Further reading

http://eclipsefp.sourceforge.net/

6.1.2 ghc-mod — Happy Haskell Programming on Emacs

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

`ghc-mod` is an enhancement of the Haskell mode on Emacs. It 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` (→ 6.3.2) 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.

Function type You can display the type/information of the function on the cursor. (new)

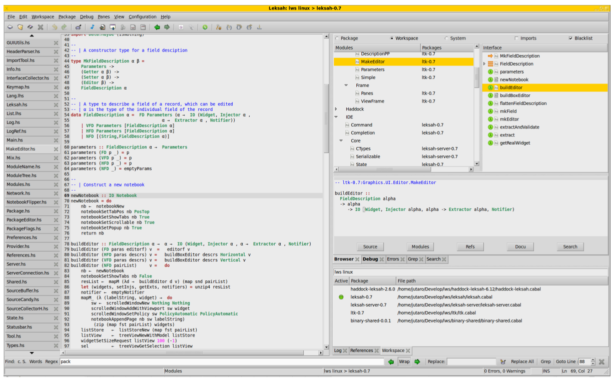
`ghc-mod` consists of code in Emacs Lisp and a sub-command in Haskell. The Emacs code executes the sub-command to obtain information about your Haskell environment. The sub-command makes use of the GHC API for that purpose. `ghc-mod` now supports "hs-source-dirs" in a cabal file and GHC 7.2.

Further reading

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

6.1.3 Leksah — The Haskell IDE in Haskell

Report by:	Jürgen Nicklisch-Franken
Participants:	Hamish Mackenzie



```
--
-- | A constructor type for a field desciption
--
type MkFieldDescription α β =
    Parameters ->
    (Getter α β) ->
    (Setter α β) ->
    (Editor β) ->
    FieldDescription α

--
-- | A type to describe a field of a record, which can be edited
-- | α is the type of the individual field of the record
data FieldDescription α =  FD Parameters (α →  IO (Widget, Injector α ,
                                         α →  Extractor α , Notifier))
    | VFD Parameters [FieldDescription α]
    | HFD Parameters [FieldDescription α]
    | NFD [(String,FieldDescription α)]

parameters :: FieldDescription α →  Parameters
parameters (FD p _) = p
parameters (VFD p _) = p
parameters (HFD p _) = p
parameters (NFD _) = emptyParams

--
-- | Construct a new notebook
--
newNotebook :: IO Notebook
newNotebook = do
    nb ←  notebookNew
    notebookSetTabPos nb PosTop
    notebookSetShowTabs nb True
    notebookSetScrollable nb True
    notebookSetPopup nb True
    return nb

buildEditor :: FieldDescription α →  α →  IO (Widget, Injector α , α →  Extractor α , Notifier)
buildEditor (FD paras editorf) v  =   editorf v
buildEditor (HFD paras descrs) v =   buildBoxEditor descrs Horizontal v
buildEditor (VFD paras descrs) v =   buildBoxEditor descrs Vertical v
buildEditor (NFD pairList)     v =   do
    nb ←  newNotebook
    notebookSetShowTabs nb False
    resList ←  mapM (λd →  buildEditor d v) (map snd pairList)
    let (widgets, setInjs, getExts, notifiers) = unzip4 resList
    notifier ←  emptyNotifier
    mapM_ (λ (labelString, widget) →  do
        sw ←  scrolledWindowNew Nothing Nothing
        scrolledWindowAddWithViewport sw widget
        scrolledWindowSetPolicy sw PolicyAutomatic PolicyAutomatic
        notebookAppendPage nb sw labelString)
         (zip (map fst pairList) widgets)
    listStore   ←  listStoreNew (map fst pairList)
    listView    ←  treeViewNewWithModel listStore
    widgetSetSizeRequest listView 100 (-1)
    sel         ←  treeViewGetSelection listView
```

Leksah is a Haskell IDE written in Haskell. It is still beta quality, but we hope we can publish the 1.0 release this year.

The project has its focus on providing a practical tool for Haskell development. Leksah has already proved its usefulness in industrial projects. We have had positive feedback and are pleased to see that a large number of people are downloading Leksah and we hope you are finding it useful.

Leksah is at a critical point in its development, as it is difficult to bring a project of this size to a success, considering we are just two developers which work on it in their rare spare time. If you can spare some time to work on part of the project, please get in touch by mailing the Leksah group or log onto IRC #leksah. If there is something you do not like about Leksah let us know and we can probably show you where to get started fixing it.

We believe that Leksah can be an important contribution for Haskell, to make its way from an academic language to a valuable tool in industry.

Further reading

http://leksah.org/

6.1.4 HEAT: The Haskell Educational Advancement Tool

Report by:	Olaf Chitil
Status:	active

See: http://www.haskell.org/communities/11-2010/html/report.html#sect6.1.4.

6.1.5 HaRe — The Haskell Refactorer

Report by:	Simon Thompson
Participants:	Huiqing Li, Chris Brown, Claus Reinke

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 sixth major release. HaRe supports full Haskell 98, and is integrated with (X)Emacs and Vim. 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. The system also contains a set of data-oriented refactorings which together transform a concrete `data` type and associated uses of pattern matching into an abstract type and calls to assorted functions. The latest snapshots support the hierarchical modules extension, but only small parts of the hierarchical libraries, unfortunately.

In order to allow users to extend HaRe themselves, HaRe includes an API for users to define their own program transformations, together with Haddock documentation. Please let us know if you are using the API.

Snapshots of HaRe are available from our webpage, as 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 there, too.

Recent developments

- HaRe 0.6, which is compatible with GHC-6.12.1, has been released; HaRe 0.6 is available on Hackage, and also downloadable from our project webpage.
- HaRe 0.6 comes with a number of new refactorings, including adding and removing fields and constructors to data-type definitions, folding and unfolding against as-patterns, merging and splitting function definitions, converting between let and where constructs, introducing pattern matching and generative folding.
- Support for automatic detection and semi-automatic elimination of duplicated code in Haskell programs is also available from HaRe 0.6.
- Support for a number of new refactorings for *parallel* Haskell have recently been added to HaRe. These include support to introduce simple divide and conquer parallelism, using the new Strategies module. The refactorings are designed to issue warnings to the user when ill-defined evaluation degrees are set, together with support for adding a threshold value.

Further reading

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

6.2 Documentation

6.2.1 Haddock

Report by:	David Waern
Status:	experimental, maintained

Haddock is a widely used documentation-generation tool for Haskell library code. Haddock generates documentation by parsing and typechecking Haskell source code directly and including documentation supplied by the programmer in the form of specially-formatted comments in the source code itself. Haddock has direct support in Cabal (→ 6.8.1), and is used to generate the documentation for the hierarchical libraries that come with GHC, Hugs, and nhc98 (http://www.haskell.org/ghc/docs/latest/html/libraries) as well as the documentation on Hackage.

The latest release is version 2.9.4, released October 3 2011.

Recent changes:

- Support for GHC 7.2 and Alex 3.x
- New –qual flag for qualification of names
- Print doc coverage information to stdout
- Speed up generation of index
- Various bug fixes

Future plans

- Although Haddock understands many GHC language extensions, we would like it to understand all of them. Currently there are some constructs you cannot comment, like GADTs and associated type synonyms.
- Error messages is an area with room for improvement. We would like Haddock to include accurate line numbers in markup syntax errors.
- On the HTML rendering side we want to make more use of Javascript in order to make the viewing experience better. The frames-mode could be improved this way, for example.
- Finally, the long term plan is to split Haddock into one program that creates data from sources, and separate backend programs that use that data via the Haddock API. This will scale better, not requiring adding new backends to Haddock for every tool that needs its own format.

Further reading

- Haddock's homepage: http://www.haskell.org/haddock/
- Haddock's developer Wiki and Trac: http://trac.haskell.org/haddock
- Haddock's mailing list: haddock@projects.haskell.org

6.2.2 Hoogle

Report by:	Neil Mitchell
Status:	stable

Hoogle is an online Haskell API search engine. It searches the functions in the various libraries, both by name and by type signature. When searching by name, the search just finds functions which contain that name as a substring. However, when searching by types it attempts to find any functions that might be appropriate, including argument reordering and missing arguments. The tool is written in Haskell, and the source code is available online. Hoogle is available as a web interface, a command line tool, and a lambdabot plugin.

Hoogle has seen significant revisions in the last few months. Hoogle can now search all of Hackage (→ 6.8.1), and has a brand new look and feel, including instant results as you type. Work continues improving the performance and quality of the results.

Further reading

http://haskell.org/hoogle

6.2.3 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 1.17, so there has not been a new release since the last report. Development repository and bug tracker are on GitHub. There are still plans for a rewrite of lhs2TEX with the goal of cleaning up the internals and making the functionality of lhs2TEX available as a library.

Further reading

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

6.3 Testing and Analysis

6.3.1 shelltestrunner

Report by:	Simon Michael
Status:	stable

shelltestrunner was first released in 2009, inspired by the test suite in John Wiegley's ledger project. It is a command-line tool for doing repeatable functional testing of command-line programs or shell commands. It reads simple declarative tests specifying a command, some input, and the expected output, error output and exit status. Tests can be run selectively, in parallel, with a timeout, in color, and/or with differences highlighted.

In the last six months, shelltestrunner has had three releases (1.0, 1.1, 1.2) and acquired a home page. Projects using it include hledger, yesod, berp, and eddie. shelltestrunner is free software released under GPLv3+ from Hackage or http://joyful.com/shelltestrunner.

Further reading

http://joyful.com/repos/shelltestrunner

6.3.2 HLint

Report by:	Neil Mitchell
Status:	stable

HLint is a tool that reads Haskell code and suggests changes to make it simpler. For example, if you call `maybe foo id` it will suggest using `fromMaybe foo` instead. HLint is compatible with almost all Haskell extensions, and can be easily extended with additional hints.

There have been numerous feature improvements since the last HCAR, including features to detect duplicated code within a module. HLint can be tried online within hpaste.org.

Further reading

http://community.haskell.org/~ndm/hlint/

6.3.3 hp2any

Report by:	Patai Gergely
Status:	experimental

This project was born during the 2009 Google Summer of Code under the name "Improving space profiling experience". The name **hp2any** covers a set of tools and libraries to deal with heap profiles of Haskell programs. At the present moment, the project consists of three packages:

- `hp2any-core`: a library offering functions to read heap profiles during and after run, and to perform queries on them.
- `hp2any-graph`: an OpenGL-based live grapher that can show the memory usage of local and remote processes (the latter using a relay server included in the package), and a library exposing the graphing functionality to other applications.
- `hp2any-manager`: a GTK application that can display graphs of several heap profiles from earlier runs.

The project also aims at replacing `hp2ps` by reimplementing it in Haskell and possibly adding new output formats. The manager application shall be extended to display and compare the graphs in more ways, to export them in other formats and also to support live profiling right away instead of delegating that task to `hp2any-graph`.

Recently, the **hp2any** project joined forces with `hp2pretty`, which resulted in increased performance in the core library.

Further reading

- http://www.haskell.org/haskellwiki/Hp2any
- http://code.google.com/p/hp2any/
- http://gitorious.org/hp2pretty

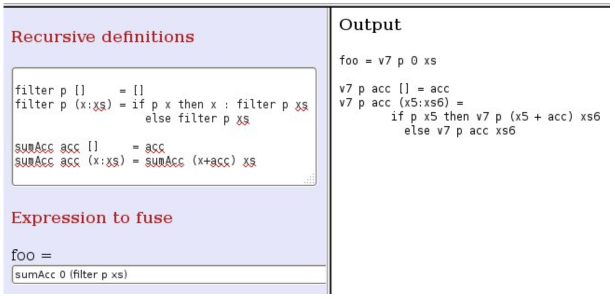
6.4 Optimization

6.4.1 HFusion

Report by:	Facundo Dominguez
Participants:	Alberto Pardo
Status:	experimental

HFusion is an experimental tool for optimizing Haskell programs. The tool performs source to source transformations by the application of a program transformation technique called *fusion*. The aim of fusion is to reduce memory management effort by eliminating the intermediate data structures produced in function compositions. It is based on an algebraic approach where functions are internally represented in terms of a recursive program scheme known as *hylomorphism*.

We offer a web interface to test the technique on user-supplied recursive definitions and HFusion is also available as a library on Hackage. The last improvement to HFusion has been to accept as input an expression containing any number of compositions, returning the expression which results from applying fusion to all of them. Compositions which cannot be handled by HFusion are left unmodified.



```
filter p []     = []
filter p (x:xs) = if p x then x : filter p xs
                  else filter p xs

sumAcc acc []     = acc
sumAcc acc (x:xs) = sumAcc (x+acc) xs
```

foo =

```
sumAcc 0 (filter p xs)
```

```
foo = v7 p 0 xs

v7 p acc [] = acc
v7 p acc (x5:xs6) =
        if p x5 then v7 p (x5 + acc) xs6
          else v7 p acc xs6
```

In its current state, HFusion is able to fuse compositions of general recursive functions, including primitive recursive functions (like dropWhile or insertions in binary search trees), functions that make recursion over multiple arguments like zip, zipWith or equality predicates, mutually recursive functions, and (with some limitations) functions with accumulators like foldl. In general, HFusion is able to eliminate intermediate data structures of regular data types (sum-of-product types plus different forms of generalized trees).

Further reading

- HFusion publications: http://www.fing.edu.uy/inco/proyectos/fusion
- HFusion web interface: http://www.fing.edu.uy/inco/proyectos/fusion/tool
- HFusion on Hackage: http://hackage.haskell.org/package/hfusion

6.4.2 Optimizing Generic Functions

Report by:	José Pedro Magalhães
Participants:	Johan Jeuring, Andres Löh
Status:	actively developed

Datatype-generic programming increases program reliability by reducing code duplication and enhancing reusability and modularity. Several generic programming libraries for Haskell have been developed in the past few years. These libraries have been compared in detail with respect to expressiveness, extensibility, typing issues, etc., but performance comparisons have been brief, limited, and preliminary. It is widely believed that generic programs run slower than hand-written code.

At Utrecht University we are looking into the performance of different generic programming libraries and how to optimize them. We have confirmed that generic programs, when compiled with the standard optimization flags of the Glasgow Haskell Compiler (GHC), are substantially slower than their hand-written counterparts. However, we have also found that advanced optimization capabilities of GHC, such as inline pragmas and rewrite rules, can be used to further optimize generic functions, often achieving the same efficiency as hand-written code.

We are continuing our research in this topic and hope to provide more information in the near future.

Further reading

http://dreixel.net/research/pdf/ogie.pdf

6.5 Boilerplate Removal

6.5.1 A Generic Deriving Mechanism for Haskell

Report by:	José Pedro Magalhães
Participants:	Atze Dijkstra, Johan Jeuring, Andres Löh, Simon Peyton Jones
Status:	actively developed

Haskell's deriving mechanism supports the automatic generation of instances for a number of functions. The Haskell 98 Report only specifies how to generate instances for the Eq, Ord, Enum, Bounded, Show, and Read classes. The description of how to generate instances is largely informal. As a consequence, the portability of instances across different compilers is not guaranteed. Additionally, the generation of instances imposes restrictions on the shape of datatypes, depending on the particular class to derive.

We have developed a new approach to Haskell's deriving mechanism, which allows users to specify how to derive arbitrary class instances using standard datatype-generic programming techniques. Generic functions, including the methods from six standard Haskell 98 derivable classes, can be specified entirely within Haskell, making them more lightweight and portable.

We have implemented our deriving mechanism together with many new derivable classes in UHC (→ 3.3) and GHC. The implementation in GHC has a more convenient syntax; consider enumeration:

```
class GEnum a where
  genum :: [a]
  default genum :: (Representable a,
                    Enum' (Rep a))
                  ⇒ [a]
  genum = map to enum'
```

The *Enum'* and *GEnum* classes are defined by the generic library writer. The end user can then give instances for his/her datatypes without defining an implementation:

```
instance (GEnum a) ⇒ GEnum (Maybe a)
instance (GEnum a) ⇒ GEnum [a]
```

These instances are empty, and therefore use the (generic) default implementation. This is as convenient as writing **deriving** clauses, but allows defining more generic classes. This implementation relies on the new functionality of default signatures, like in *genum* above, which are like standard default methods but allow for a different type signature.

Further reading

http://www.haskell.org/haskellwiki/Generics

6.6 Code Management

6.6.1 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 2.5.2, was in March 2011. The Darcs 2.5.x line provides faster repository-local operations, and faster record with long patch histories, among other bug fixes and features. The most recent version adds compatibility with Haskell Platform 2011.2.0.0.

We are currently working on releasing Darcs 2.8, which will include Alexey Levan's 2010 Google Summer of Code work on optimised darcs get (using the "optimize –http" command) and a few refinements to Adolfo Builes' cache reliability work. The Darcs 2.8 release is planned to include a faster and more human-readable annotate command.

Meanwhile, we are happy to have been able to participate in the Google Summer of Code 2011 (as part of Haskell.org). We had two projects this year, one to develop a a bidirectional bridge between Darcs and Git (and potentially other VCSs), and the other to do some new exploratory work on primitive patch types for a future Darcs 3. The bridge project will improve collaboration between Darcs and Git users, allowing each to contribute to projects hosted in the other's VCS of choice. The primitive patches work will allow us to implement some ideas we have been discussing in the Darcs team in recent months, in particular, separation of file dentifiers from file names and the separation of on-disk patch contents from their in-memory representation. Making a prototype implementation of these ideas will give us a better idea how feasible they are in practice and help us to identify the technical difficulities that may be lurking around the corner. Both projects were succesful; see below for their respective wrap-ups and prototypes.

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://web.mornfall.net/blog/soc_reloaded:_outcomes.html

6.6.2 DarcsWatch

Report by:	Joachim Breitner
Status:	working

See: http://www.haskell.org/communities/05-2011/html/report.html#sect5.6.3.

6.6.3 darcsden

Report by:	Simon Michael
Participants:	Alex Suraci, Simon Michael, Scott Lawrence, Daniel Patterson, Daniel Goran
Status:	beta, low activity

http://darcsden.com is a free Darcs (→ 6.6.1) repository hosting service, similar to `patch-tag.com` or (in essence) `github`. The darcsden software is also available (on darcsden) so that anyone can set up a similar service. darcsden is available under BSD license and was created by Alex Suraci.

Alex keeps the service running and fixes bugs, but is mostly focussed on other projects. darcsden has a clean UI and codebase and is a viable hosting option for smaller projects despite occasional glitches.

The last Hackage release was in 2010. Other committers have been submitting patches, and the darcsden software is close to becoming a just-works installable darcs web ui for general use.

Further reading

http://darcsden.com

6.6.4 darcsum

Report by:	Simon Michael
Status:	occasional development; suitable for daily use

darcsum is an emacs add-on providing an efficient, pcl-cvs-like interface for the Darcs revision control system (→ 6.6.1). It is especially useful for reviewing and recording pending changes.

Simon Michael took over maintainership in 2010, and tried to make it more robust with current Darcs. The tool remains slightly fragile, as it depends on Darcs' exact command-line output, and needs updating when that changes. Dave Love has contributed a large number of cleanups. darcsum is available under the GPL version 2 or later from http://joyful.com/darcsum.

In the last six months darcsum acquired a home page, but there has been little other activity. We are looking for a new maintainer for this useful tool.

Further reading

http://joyful.com/darcsum/

6.6.5 Improvements to Cabal's Test Support

Report by:	Thomas Tuegel
Participants:	Johan Tibell (Mentor)
Status:	active development

See: http://www.haskell.org/communities/11-2010/html/report.html#sect7.2.

6.6.6 `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`, `cabal`, and `cabal-dev`.

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 GHC 7.2.

Further reading

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

6.6.7 Hackage-Debian

Report by:	Marco Gontijo
Status:	unconcluded

Hackage-Debian is a tool for creating a Debian repository with all, or almost all, of the packages in Hackage. It is highly based on the debian available at http://hackage.haskell.org/package/debian. It should build a snapshot of the Hackage database and then track each new package added to build it on demand. It is still under development, but the first release should be announced soon.

A limitation of the first version being developed is that it only builds the latest version of each library. So, if a library depends on an older version of another library, it will not be built. This is the reason why it does not build all packages, but almost all of them.

Also, the first version will only deal with libraries, but there are plans to also build programs.

The darcs repository for both hackage-debian and the modified version of the debian package that it uses are available at http://marcot.eti.br/darcs/hackage-debian and http://marcot.eti.br/darcs/haskell-debian.

6.7 Interfacing to other Languages

6.7.1 HSFFIG

Report by:	Dmitry Golubovsky
Status:	release

See: http://www.haskell.org/communities/11-2010/html/report.html#sect6.6.1.

6.8 Deployment

6.8.1 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

We have had two successful Google Summer of Code projects on Cabal this year. Sam Anklesaria worked on a “cabal repl” feature to launch an interactive GHCi session with all the appropriate pre-processing and context from the project’s `.cabal` file. Mikhail Glushenkov worked on a feature so that “cabal install” can build independent packages in parallel (not to be confused with building modules within a package in parallel). The code from both projects is available and they are awaiting integration into the main Cabal repository, which we expect to happen over the course of the next few months.

The “cabal test” feature which was developed as a GSoC project last summer has matured significantly in the last 6 months, thanks to continuing effort from Thomas Tuegel and Johan Tibell. The basic test interface will be ready to use in the next release, and there has been some progress on the “detailed” test interface.

The IHG is currently sponsoring some work on cabal-install. The first fruits of this work is a new dependency solver for cabal-install which is now included in the development version. The new solver can find solutions in more cases and produces more detailed error messages when it cannot find a solution. In addition, it is better about avoiding and warning about breaking existing installed packages. We also expect it to be a better basis for other features in future. For more details see the presentation by Andres Löh.

http://haskell.org/haskellwiki/HaskellImplementorsWorkshop/2011/Loeh

The last 6 months has seen significant progress on the new `hackage-server` implementation with help from many new volunteers, in particular Max Bolingbroke, but also several other people who helped at hackathons and subsequently. The IHG funded Well-Typed to improve package mirroring so that continuous nearly-live mirroring is now possible. We are also grateful to factis research GmbH who have kindly donated a VM to help the hackage developers test the new server code. We expect to do live mirroring and public beta testing using this server during the next few months.

Looking forward

Users are increasingly relying on hackage and cabal-install and are increasingly frustrated by dependency problems. Solutions to the variety of problems do exist. It will however take sustained effort to solve them. The good news is that there is the realistic prospect of the new hackage-server being ready in the not too distant future with features to help monitor and encourage package quality, and the recent work on cabal-install should reduce the frustration level somewhat.

The last 6 months has seen a good upswing in the number of volunteers spending their time on cabal and hackage, so much so that a clear bottleneck is patch review and integration bandwidth. A similar issue is that many of the long standing bugs and feature requests require significant refactoring work which many volunteers feel reluctant or unable to do. Assistance in these areas would be very valuable indeed.

We would like to encourage people considering contributing to join the `cabal-devel` mailing list so that we can increase development discussion and improve collaboration. 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.

Further reading

- Cabal homepage: http://www.haskell.org/cabal
- Hackage package collection: http://hackage.haskell.org/
- Bug tracker: http://hackage.haskell.org/trac/hackage/

6.8.2 Capri

Report by:	Dmitry Golubovsky
Status:	experimental

See: http://www.haskell.org/communities/11-2010/html/report.html#sect6.7.2.

7 Libraries

7.1 Processing Haskell

7.1.1 The Neon Library

Report by:	Jurriaan Hage

See: http://www.haskell.org/communities/11-2010/html/report.html#sect8.1.1.

7.2 Parsing and Transforming

7.2.1 The grammar-combinators Parser Library

Report by:	Dominique Devriese
Status:	partly functional

See: http://www.haskell.org/communities/11-2010/html/report.html#sect8.2.1.

7.2.2 epub-metadata

Report by:	Dino Morelli
Status:	stable, actively developed

Library for parsing and manipulating ePub files and OPF package data. An attempt has been made here to very thoroughly implement the OPF Package Document specification.

epub-metadata is available from Hackage, the Darcs repository below, and also in binary form for Arch Linux through the AUR.

See also epub-tools (→ 8.8.10).

Further reading

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

7.2.3 Utrecht Parser Combinator Library: uu-parsinglib

Report by:	Doaitse Swierstra
Status:	actively developed

The previous extension for recognizing merging parsers was generalized so now any kind of applicative and monadic parsers can be merged in an interleaved way. As an example take the situation where many different programs write log entries into a log file, and where each log entry is uniquely identified by a transaction number (or process number) which can be used to distinguish them. E.g., assume that each transaction consists of an *a*, a *b* and a *c* action, and that a digit is used to identify the individual actions belonging to the same transaction; the individual transactions can now be recognized by the parser:

$$\begin{array}{l} pABC :: Grammar\ String \\ pABC = (\lambda a\ d \rightarrow d : a) <\$> pA <\!\!*\!\!> (pDigit' \gg\!= \\ \qquad\qquad \lambda d \rightarrow pB\ \ *\!\!>\ mkGram\ (pSym\ d)\ \ *\!\!> \\ \qquad\qquad\qquad\ \ pC\ \ *\!\!>\ mkGram\ (pSym\ d) \\ \qquad\qquad) \end{array}$$

Now running many merged instances of this parser on the input returns the list of first lines prefixed by their number:

```
run (pmMany(pABC)) "a2a1b1b2c2a3b3c1c3"
 Result: ["2a","1a","3a"]
```

Furthermore the library was provided with many more examples in two modules in the *Demo* directory.

Features

- Much simpler internals than the old library (http://haskell.org/communities/05-2009/html/report.html#sect5.5.8).
- 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 makes 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

Since the part dealing with merging is relatively independent of the underlying parsing machinery we may split this off into a separate package. This will enable us also to make use of a different parsing engines when combining parsers in a much more dynamic way. In such cases we want to avoid too many static analyses.

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 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.

The old library in the *uulib* package stays stable, and can continue to be used. A few changes were needed in order to make it compile with GHC 7.2.

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⟩. There is a low volume, moderated mailing list which was moved to ⟨parsing@lists.science.uu.nl⟩ (see also http://www.cs.uu.nl/wiki/bin/view/HUT/ParserCombinators).

7.2.4 Regular Expression Matching with Partial Derivatives

Report by:	Martin Sulzmann
Participants:	Kenny Zhuo Ming Lu
Status:	stable

We are still improving the performance of our matching algorithms. The latest implementation can be downloaded via hackage.

Further reading

- http://hackage.haskell.org/package/regex-pderiv
- http://sulzmann.blogspot.com/2010/04/regular-expression-matching-using.html

7.2.5 regex-applicative

Report by:	Roman Cheplyaka
Status:	active development

regex-applicative is aimed to be an efficient and easy to use parsing combinator library for Haskell based on regular expressions.

Regular expressions have Perl-like (left-biased) semantics to satisfy most of the daily regex needs, but also allow longest matching prefix search useful for lexical analysis.

For example, the following code finds filename extensions:

```
import Text.Regex.Applicative

getExtension :: String -> Maybe String
getExtension str =
    str =~
        many anySym *>
        sym '.'     *>
        many anySym
```

More examples can be found on the wiki.

Further reading

- http://hackage.haskell.org/package/regex-applicative
- http://github.com/feuerbach/regex-applicative

7.3 Mathematical Objects

7.3.1 normaldistribution: Minimum Fuss Normally Distributed Random Values

Report by:	Björn Buckwalter
Status:	stable

Normaldistribution is a new package that lets you produce normally distributed random values with a minimum of fuss. The API builds upon, and is largely analogous to, that of the Haskell 98 Random module (more recently *System.Random*). Usage can be as simple as: $sample \leftarrow normalIO$. For more information and examples see the package description on Hackage.

Further reading

http://hackage.haskell.org/package/normaldistribution

7.3.2 dimensional: Statically Checked Physical Dimensions

Report by:	Björn Buckwalter
Status:	active, stable core with experimental extras

Dimensional is a library providing data types for performing arithmetics with physical quantities and units. Information about the physical dimensions of the quantities/units is embedded in their types, and the validity of operations is verified by the type checker at compile time. The boxing and unboxing of numerical values as quantities is done by multiplication and division with units. The library is designed to, as far as is practical, enforce/encourage best practices of unit usage within the frame of the SI. Example:

$$\begin{aligned}&d :: Fractional\ a \Rightarrow Time\ a \rightarrow Length\ a\\&d\ t = a\ /\ \_2 * t\ \hat{}\ pos2\\&\quad \mathbf{where}\ a = 9.82 *\tilde{}\ (meter\ /\ second\ \hat{}\ pos2)\end{aligned}$$

The dimensional library is stable with units being added on an as-needed basis. The primary documentation is the literate Haskell source code. The wiki on the project web site has several usage examples to help with getting started.

Ongoing experimental work includes:

- Support for user-defined dimensions and a proof-of-concept implementation of the CGS system of units.
- dimensional-vectors — a rudimentary linear algebra library which statically tracks the sizes of vectors and matrices as well as the physical dimensions of their elements on a per element basis, disallowing non-sensical operations. This library makes it very difficult to accidentally implement, e.g., a Kalman filter incorrectly. My work on dimensional-vectors is need-driven and tends to occur in spurts.
- dimensional-experimental — a library in heavy flux of which the most interesting feature is probably automatic differentiation of functions involving physical quantities. Example:

  $v :: \mathit{Fractional}\ a \Rightarrow \mathit{Time}\ a \rightarrow \mathit{Velocity}\ a$
  $v\ t = \mathit{diff}\ \ d\ t$

The core library, dimensional, can be installed off Hackage using cabal. The experimental packages can be cloned off of Github.

Dimensional relies on *numtype* for type-level integers (e.g., *pos2* in the above example), *ad* for automatic differentiation, and *HList* (→ 7.4.1) for type-level vector and matrix representations.

Further reading

- http://dimensional.googlecode.com
- https://github.com/bjornbm/dimensional-vectors
- https://github.com/bjornbm/dimensional-experimental

7.3.3 AERN-Real and Friends

Report by:	Michal Konečný
Participants:	Jan Duracz
Status:	experimental, actively developed

AERN stands for Approximating Exact Real Numbers. We are developing a family of libraries that will provide:

- a reliable and fast arbitrary precision correctly rounded **interval arithmetic**, including both standard and inverted intervals with Kaucher arithmetic
- arbitrary precision arithmetic of **interval polynomials** and **polynomial intervals** to
  - automatically reduce overestimations in interval computations
  - efficiently support validated numerical integration
  - automatically decide many inequalities and interval inclusions with non-linear and elementary functions that occur in numerical theorem proving and specifically in the verification of numerical programs
- a type class hierarchy for validated and exact computation, featuring
  - standard mathematical structures such as posets and lattices extended to take account of rounding errors and partially decided relations such as equality
  - separate treatment of numerical order and interval refinement order
  - ability to increase computational effort to reduce the effect of rounding and partiality, converging to no rounding and total relations with infinite effort
  - extensive set of QuickCheck properties for each type class, enabling automatic checking of, e.g., algebraic properties such as associativity extended to take account of rounding
- a framework for distributed query-driven lazy dataflow exact numerical computation with tidy exact semantics based on Domain Theory

There are stable older versions of the libraries on Hackage but these lack the type classes described above.

We are currently in the process of redesigning and rewriting the libraries from scratch. Out of the newly designed code we recently released libraries featuring

- the type classes for approximate real number operations
- correctly rounded real interval arithmetic with Double endpoints

A release of interval arithmetic with MPFR endpoints is planned as soon as a solution is found for an easier installation of the hmpfr package. (Currently one has to compile a ghc without gmp to use hmpfr.)

We have made progress on implementing polynomial intervals with a core written in C but have suspended the development until we finish a Haskell-only implementation of an arithmetic of interval polynomials (ie polynomials with interval coefficients). We are likely to use interval polynomials as endpoints for polynomial intervals when the work on polynomial intervals is resumed.

The development files now include demos that apply interval polynomials on validated simulation of selected ODE IVPs and hybrid systems.

All AERN development is open and we welcome contributions and new developers.

Further reading

http://code.google.com/p/aern/

7.3.4 Paraiso

Report by:	Takayuki Muranushi
Status:	active development

Paraiso is a domain-specific language (DSL) embedded in Haskell, aimed at generating explicit type of partial differential equations solving programs, for accelerated and/or distributed computers. Equations for fluids, plasma, general relativity, and many more falls into this category. This is still a tiny domain for a computer scientist, but large enough that an astrophysicist (I am) might spend even his entire life in it.

In Paraiso we can describe equation-solving algorithms in mathematical, simple notation using *builder monads*. At the moment it can generate programs for multicore CPUs as well as single GPU, and tune their performance via automated benchmarking and genetic algorithms. The experiment is under way; the fluid simulator I am using is 464 lines in Haskell. So far, Paraiso has tried more than 117'000 different implementations of this single algorithm, each being about 10'000 lines of CUDA program. The best one found so far is 33.4 times faster than the initial guess, and twice faster than the hand-tuned implementation.

Anyone can get Paraiso from hackage (http://hackage.haskell.org/package/Paraiso) or github (https://github.com/nushio3/Paraiso).

The next big challenge is to make Paraiso generate distributed computations.

Further reading

http://paraiso-lang.org/wiki/

7.4 Data Types and Data Structures

7.4.1 HList — A Library for Typed Heterogeneous Collections

Report by:	Oleg Kiselyov
Participants:	Ralf Lämmel, Keean Schupke, Gwern Branwen

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 (→5.4.2), typed EDSL of attribute grammars, and in HaskellDB.

The October 2011 version of HList library has many changes, mainly related to deprecating `TypeCast` (in favor of `~`) and getting rid of overlapping instances. The only use of OverlappingInstances is in the implementation of the generic type equality predicate `TypeEq`. We plan to remove even that remaining single occurrence. The code works with GHC 7.0.4.

Future plans include the implementation of TypeEq without resorting to overlapping instances (so, HList will be overlapping-free), and moving towards type functions and expressive kinds.

Further reading

- HList: http://okmij.org/ftp/Haskell/types.html#HList
- OOHaskell: http://homepages.cwi.nl/~ralf/OOHaskell/

7.4.2 Persistent

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

Persistent is a type-safe data store interface for Haskell. Haskell has many different database bindings available. However, most of these have little knowledge of a schema and therefore do not provide useful static guarantees. They also force database-dependent interfaces and data structures on the programmer.

There are Haskell specific data stores such as acid-state that get around these flaws. This allows one to easily store any Haskell type and have type-safe interactions with data. However, the use case is limited to in memory storage without replication, and they aren't designed to interface with other programming languages.

Persistent maintains much of the advantage of using native Haskell data types — you store and retrieve normal Haskell records, and your queries are also type-safe — they must match the schema. However, Persistent lets you persist your data to a battle tested database of your choice that is well optimized for your problem domain. Persistent is backend agnostic, and there are currently interfaces to Sqlite, Postgresql, and MongoDB.

Since the last report, Persistent has undergone an internal re-write and major API changes. The MongoDB backend has been polished and works out of the box with the Yesod web framework. Here is a quick example of the new Persistent query language:

$$\mathit{selectList}\ [\mathit{PersonFirstName} == . \texttt{"Simon"},\\ \mathit{PersonLastName} == . \texttt{"Jones"}]\ [\,]$$

Future plans

There are 3 main directions for Persistent:
- Improvements that work across all Persistent backends (example: better application-level joins)
- Better database-specific integration (example: better SQL joins)
- Adding more database backends

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://yesodweb.com/book/persistent

7.5 Generic and Type-Level Programming

7.5.1 Unbound

Report by:	Brent Yorgey
Participants:	Stephanie Weirich, Tim Sheard
Status:	actively maintained

Unbound is a domain-specific language and library for working with binding structure. Implemented on top of the RepLib generic programming framework, it automatically provides operations such as alpha equivalence, capture-avoiding substitution, and free variable calculation for user-defined data types, requiring only a tiny bit of boilerplate on the part of the user. It features a simple yet rich combinator language for binding specifications, including support for pattern binding, type annotations, recursive binding, nested binding, and multiple atom types.

Since the last HCAR, a new version of Unbound has been released, adding support for several set-like binding strategies (where the order of bound variables does not matter) and for GADTs which do not use existential quantification.

Further reading

- http://byorgey.wordpress.com/2011/08/24/unbound-now-supports-set-binders-and-gadts/
- http://byorgey.wordpress.com/2011/03/28/binders-unbound/
- http://hackage.haskell.org/package/unbound
- http://code.google.com/p/replib/

7.5.2 FlexiWrap

Report by:	Iain Alexander
Status:	experimental

A library of flexible newtype wrappers which simplify the process of selecting appropriate typeclass instances, which is particularly useful for composed types.

Version 0.1.0 has been released on Hackage, providing support for a more comprehensive range of typeclasses when wrapping simple values, and some documentation. Work is still ongoing to flesh out the typeclass instances available and improve the documentation.

7.5.3 Generic Programming at Utrecht University

Report by:	José Pedro Magalhães
Participants:	Johan Jeuring, Sean Leather
Status:	actively developed

One of the research themes investigated within the Software Technology Center in the Department of Information and Computing Sciences at Utrecht University is generic programming. Over the last 15 years, we have played a central role in the development of generic programming techniques, languages, and libraries.

Currently we maintain a number of generic programming libraries and applications. We report most of them in this entry; `emgm` was reported on before (http://haskell.org/communities/05-2009/html/report.html#sect5.9.3), and our generic deriving mechanism has its own entry (→ 6.5.1).

`multirec` This library represents datatypes uniformly and grants access to sums (the choice between constructors), products (the sequence of constructor arguments), and recursive positions. Families of mutually recursive datatypes are supported. Functions such as *map*, *fold*, *show*, and equality are provided as examples within the library. Using the library functions on your own families of datatypes requires some boilerplate code in order to instantiate the framework, but is facilitated by the fact that `multirec` contains Template Haskell code that generates these instantiations automatically.

The `multirec` library can also be used for type-indexed datatypes. As a demonstration, the `zipper` library is available on Hackage. With this datatype-generic zipper, you can navigate values of several types.

The latest version available on Hackage includes limited support for datatype compositions; we are still planning to extend the library with support for parameterized datatypes.

`regular` While `multirec` focuses on support for mutually recursive regular datatypes, `regular` supports only single regular datatypes. The approach used is similar to that of `multirec`, namely using type families to encode the pattern functor of the datatype to represent generically. There have been no major releases of the `regular` or `regular-extras` packages on Hackage since the last report. The current versions provide a number of typical generic functions, but also some less well-known but useful functions: deep *seq*, `QuickCheck`'s *arbitrary* and *coarbitrary*, and `binary`'s *get* and *put*.

`instant-generics` Using type families and type classes in a way similar to `multirec` and `regular`, `instant-generics` is yet another approach to generic programming, supporting a large variety of datatypes and allowing the definition of type-indexed datatypes. It was first described by Chakravarty et al., and forms the basis of one of our rewriting libaries. It is available on Hackage.

`syb` Scrap Your Boilerplate (`syb`) has been supported by GHC since the 6.0 release. This library is based on combinators and a few primitives for type-safe casting and processing constructor applications. It was originally developed by Ralf Lämmel and Simon Peyton Jones. Since then, many people have contributed with research relating to `syb` or its applications.

Since `syb` has been separated from the `base` package, it can now be updated independently of GHC. We have recently released version 0.3 on Hackage, which has some minor extensions and fixes.

Annotations We presented two applications of generic annotations at the Workshop on Generic Programming 2010: selections and storage. In the former we use annotations at every recursive position of a datatype to allow for inserting position information automatically. This allows for informative parsing error messages without the need for explicitly changing the datatype to contain position information. In the latter we use the annotations as pointers to locations in the heap, allowing for transparent and efficient data structure persistency on disk.

Rewriting We also maintain two libraries for generic rewriting: a simple, earlier library based on `regular`, and the guarded rewriting library, based on `instant-generics`. The former allows for rewriting only on regular datatypes, while the latter supports more datatypes and also rewriting rules with preconditions.

We also continue to look at benchmarking and improving the performance of different libraries for generic programming (→ 6.4.2).

Further reading

http://www.cs.uu.nl/wiki/GenericProgramming

7.6 User Interfaces

7.6.1 Gtk2Hs

Report by:	Axel Simon
Participants:	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

In a heroic effort, Duncan Coutts has adjusted Gtk2Hs and its build system to run with GHC 7.X compilers. A release 0.12.1 was the result of this effort which, however, was only announced on the Gtk2Hs website. Since then a few but important bugs have been fixed, amongst them one relating to slow Cairo drawing. These bug fixes are now in the current 0.12.2 release.

Further reading

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

7.7 Graphics

7.7.1 Assimp

Report by:	Joel Burget
Status:	actively developed

Assimp is a set of bindings to the Assimp Open Asset Import Library. This library can import many different types of 3D models for use in graphics. The full list of formats is available at the project website (linked below) and at the git repo for the project. Assimp is being developed alongside the Cologne ray tracer (→ 8.4.5) but could be useful in any 3D graphics project.

Further reading

- https://github.com/joelburget/assimp
- http://assimp.sourceforge.net

7.7.2 plot/plot-gtk

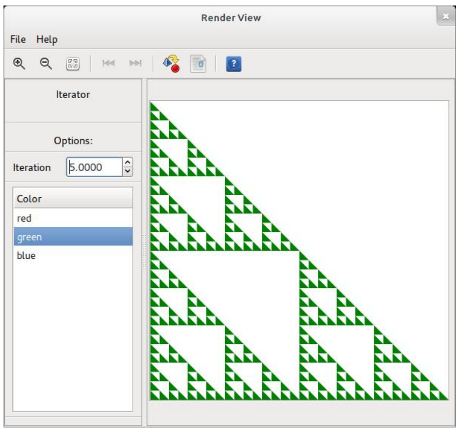
Report by:	Vivian McPhail
Status:	active development

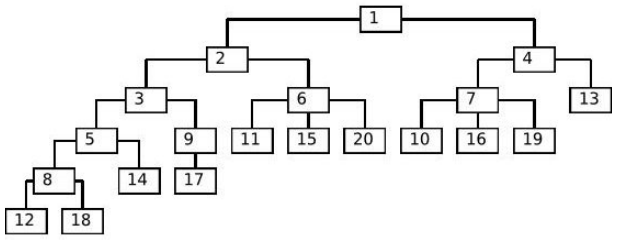
See: http://www.haskell.org/communities/05-2011/html/report.html#sect6.7.2.

7.7.3 Craftwerk

Report by:	Malte Harder
Participants:	Jannis Harder
Status:	active development

Craftwerk is a 2D vector graphic library. The motivation was to have a graphic library that is able to generate output which can be embedded into LaTeX as well as support for rendering with Cairo. Thus the library separates the graphic's data structure from any context dependency and the aim is to support various drivers. Currently a driver for output with the TikZ package (http://sourceforge.net/projects/pgf/) for LaTeX is available. Using the additional `craftwerk-cairo` and `craftwerk-gtk` packages, direct rendering into PDF files or GTK widgets is possible. The `craftwerk-gtk` package also provides functions to generate simple user interfaces for interactive graphics.





Above, two examples are shown. In the first, you can see a screenshot of the GTK interface for interactive graphics showing a Sierpiński triangle, and the second is a simple example of a tree rendered with the Cairo driver. Graphics or figures can be created in a hierarchical fashion including the application of styles and decorations to subnodes. The current functionality includes almost the complete Cairo function set extended by arrow tips and a few primitives. The same function set is supported for TikZ output, and graphics generated with the two drivers match closely. Immediate development tasks are:

- Improvement of rendering speed in the Cairo driver.
- Better and unified text rendering capabilities.
- Refactoring of the UI module towards better usability.

Besides additional functionality, a long term goal is to support other drivers like Wumpus, Haha (ASCII rendering) or OpenGL. Craftwerk could also serve as an intermediate layer for libraries like `plot` or `chart` to enable LaTeX export. At the moment the library is still at a preliminary stage and the next step is a consolidation of a basic feature set. Any contributions or ideas are welcome and the latest code as well as experiments with other drivers are available on GitHub.

Further reading

- http://hackage.haskell.org/package/craftwerk-0.1
- http://mahrz.github.com/craftwerk

7.7.4 LambdaCube

Report by:	Csaba Hruska
Status:	experimental, active development

LambdaCube is a 3D rendering engine entirely written in Haskell.

The main goal of this project is to provide a modern and feature rich graphical backend for various Haskell projects, and in the long run it is intended to be a practical solution even for serious purposes. The engine uses Ogre3D's (http://www.ogre3d.org) mesh and material file format, therefore it should be easy to find or create new content for it. The code sits between the low-level C API (raw OpenGL, DirectX or anything equivalent; the engine core is graphics backend agnostic) and the application, and gives the user a high-level API to work with.

The most important features are the following:

- loading and displaying Ogre3D models
- resource management
- modular architecture

If your system has OpenGL and GLUT installed, the `lambdacube-examples` package should work out of the box. The engine is also integrated with the Bullet physics engine (→ 8.8.7), and you can find a running example in the `lambdacube-bullet` package.

Since the last update, the current version of the library saw only a few minor updates: fast serialisation (using its own binary format) and various bugfixes. Behind the scenes, we are working on a completely new version, which will provide a graphics-oriented dataflow DSL (in the same spirit as GPipe). The goal is to allow the description of complex effects without mutable variables.

In the meantime, we also built a fully functional Stunts example, which is available as a separate package.

Everyone is invited to contribute! You can help the project by playing around with the code, thinking about API design, finding bugs (well, there are a lot of them anyway), creating more content to display, and generally stress testing the library as much as possible by using it in your own projects.

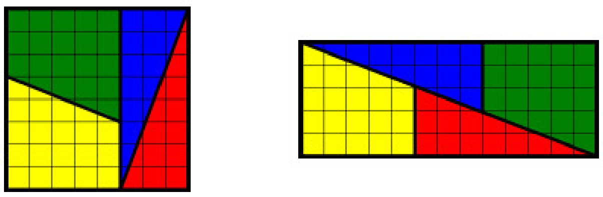
Further reading

- http://www.haskell.org/haskellwiki/LambdaCubeEngine
- http://hackage.haskell.org/package/stunts
- http://www.youtube.com/watch?v=kDu5aCGc8l4

7.7.5 diagrams

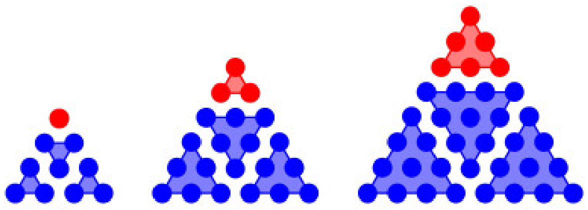
Report by:	Brent Yorgey
Participants:	Ryan Yates
Status:	active development

The diagrams library 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.



Development on the library has proceeded apace since the last HCAR, and the 0.4 release now features a comprehensive user manual as well as support for a large collection of primitive shapes, many different modes of composition, paths, cubic splines, images, text, arbitrary monoidal annotations, named subdiagrams, and more.

There is plenty more work to be done; new contributors are particularly welcome!



Future plans

Plans for the near future include a native SVG backend, improved font support, arrowheads, and improvements to the handling of named subdiagrams. Longer-term plans include support for animations, a custom Gtk application for editing diagrams, and any other awesome stuff we think of.

Further reading

- http://projects.haskell.org/diagrams
- http://code.google.com/p/diagrams/issues/list

7.7.6 ChalkBoard

Report by:	Andy Gill
Status:	suspended

ChalkBoard is a domain specific language for describing images. The language is uncompromisingly functional and encourages the use of modern functional idioms. The novel contribution of ChalkBoard is that it uses off-the-shelf graphics cards to speed up rendering of our functional description. We always intended to use ChalkBoard to animate educational videos, as well as for processing streaming videos. ChalkBoard also has an animation language, based round an applicative functor, `Active`. It has been called Functional Reactive Programming, without the reactive part!

ChalkBoard has been released on hackage, but is not actively being developed. It would be nice to port the code to HTML5, and we are happy to act as mentors for this effort.

Kevin Matlage graduated in May 2011 with an MS. His MS thesis was about the design, implementation and applications of ChalkBoard. The thesis was awarded the departmental Miller award, for best MS of the year. Congratulations Kevin!

Further reading

http://www.ittc.ku.edu/csdl/fpg/Tools/ChalkBoard

7.8 Text and Markup Languages

7.8.1 HaTeX

Report by:	Daniel Díaz
Status:	In development
Current release:	Version 3

Description

HaTeX is an implementation of LaTeX, with the aim to be a helpful tool to generate LaTeX code.

From a global sight, it's composed of:

1. The LaTeX syntax description.
2. A renderer of LaTeX code.
3. A set of combinators of LaTeX entities.
4. A monadic implementation of combinators.
5. Methods for a subset of LaTeX packages.

What is new?

The third version of HaTeX has just released, and it is a completely new implementation. Althought a lot of code is still valid, the internal representation of values has changed drastically. Now, the LaTeX code is written in an Abstract Syntax Tree (AST), via the `LaTeX` datatype.

Future plans

A near future plan is to analyze the final AST output to find possible errors in your LaTeX code, and to warn you about this. Code is already adapting to this feature.

Other new coming features are tree draws from a tree data structure, to extend the $\mathcal{AMS}$-LaTeX functionality (currently, too limited) and to implement a LaTeX code parser.

Contact

If you are someway interested in this project, please, feel free to give any kind of opinion or idea, or to ask any question you have. A good place to take contact and stay tuned is the HaTeX mailing list:

`hatex <at> projects.haskell.org`

Of course, you always can mail to the maintainer.

Further reading

- HaTeX project page: http://dhelta.net/hprojects/HaTeX

7.8.2 Haskell XML Toolbox

Report by:	Uwe Schmidt
Status:	seventh major release (current release: 9.1)

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-xmlschema.

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
- 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

In October 2011 a project has been started as part of a master thesis for an XML validator based on XML Schema. Experiences with developing the Relax-NG have shown, that such a project may be done within such a limited time. The XML picklers can be used to easily parse XML Schema an transform it into an AST. So the core work consists of developing an appropriate abstract syntax and to normalize, check and transform this AST before validating XML documents. Within this project the XML data type library already in use with Relax-NG is planned to be completed. In the current version, time and date data types are not yet supported. We expect to finish the work in March 2012.

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.

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).

8 Applications and Projects

8.1 Education

8.1.1 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-exts`. 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, but is currently unpublished.

The tool will *not* be made available through Hackage, but will be available free of use to lecturers on request. Please contact *J.Hage@uu.nl* for more information.

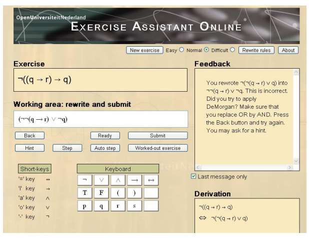
We also have a implemented graph based that computes near graph-isomorphism that seems to work really well in comparing control-flow graphs in an inexact fashion. However, it does not scale well enough in terms of computations to be included in the comparison, and is not mature enough to deal with certain easy refactorings.

Future work includes a Hare-against-Holmes bash in which Hare users will do their utmost to fool Holmes.

8.1.2 Interactive Domain Reasoners

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

The IDEAS project (at Open Universiteit Nederland and Universiteit Utrecht) aims at developing interactive domain reasoners 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 all 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 (DWO) of the Freudenthal Institute, the ActiveMath learning system (DFKI and Saarland University), and our own online exercise assistant that supports rewriting logical expressions into disjunctive normal form.

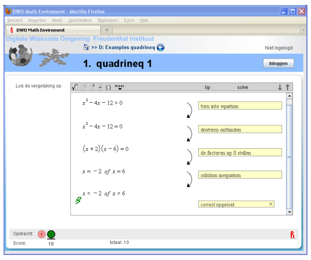


Exercise: ¬((q → r) → q)

Working area: (¬¬(q → r) ∨ ¬q)

Feedback: You rewrote ¬(¬(q → r) ∨ q) into ¬¬(q → r) ∨ ¬q. This is incorrect. Did you try to apply DeMorgan? Make sure that you replace OR by AND. Press the Back button and try again. You may ask for a hint.

Derivation:
¬((q → r) → q)
⇔ ¬(¬(q → r) ∨ q)


DWO Math Enviroment - Mozilla Firefox
Bestand Bewerken Beeld Geschiedenis Bladwijzers Extra Help
DWO Math Enviroment
Digitale Wiskunde Omgeving Freudenthal Instituut
>> D: Examples quadrineq
Niet ingelogd
1. quadrineq 1
Inloggen
tip
solve
Opdracht:
Score: 10
totaal: 10

Los de vergelijking op.

$x^2 - 4x - 12 > 0$ — turn into equation
$x^2 - 4x - 12 = 0$ — drieterm ontbinden
$(x+2)(x-6) = 0$ — de factoren op 0 stellen
$x = -2$ of $x = 6$ — solution inequation
$x < -2$ of $x > 6$ — correct opgelost

We are adding support for more exercise types, mainly at the level of high school mathematics. For example, our tool now covers simplifying expressions with exponents, rational equations, and derivatives. We have investigated how users can interleave solving different parts of exercises. Recently, we have focused on designing a functional programming tutor. This tool lets you practice introductory functional programming exercises. We are investigating how we can add testing to the tutor, and how we can let teachers configure the tutor for particular programming exercises. This is ongoing research.

The feedback services are available as a Cabal source package. The latest release is version 1.0 from September 1, 2011.

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.
- Bastiaan Heeren and Johan Jeuring. Interleaving Strategies. Conference on Intelligent Computer Mathematics, Mathematical Knowledge Management (MKM 2011).
- Johan Jeuring, Alex Gerdes, and Bastiaan Heeren. A Programming Tutor for Haskell. To appear in Lecture Notes Central European School on Functional Programming, (CEFP 2011). Try our tutor at http://ideas.cs.uu.nl/ProgTutor/.

8.2 Data Management and Visualization

8.2.1 HaskellDB

Report by:	Justin Bailey
Status:	active development

See: http://www.haskell.org/communities/11-2010/html/report.html#sect9.2.1.

8.2.2 Pandoc

Report by:	John MacFarlane
Participants:	Andrea Rossato, Peter Wang, Paulo Tanimoto, Eric Kow, Luke Plant, Justin Bogner, Paul Rivier, Nathan Gass, Puneeth Chaganti, Josef Svenningsson, Etienne Millon, Joost Kremers
Status:	active development

Pandoc aspires to be the swiss army knife of text markup formats: it can read markdown and (with some limitations) HTML, LaTeX, Textile, and reStructuredText, and it can write markdown, reStructuredText, HTML, DocBook XML, OpenDocument XML, ODT, RTF, groff man, MediaWiki markup, GNU Texinfo, LaTeX, ConTeXt, EPUB, Textile, Emacs org-mode, Slidy, and S5. Pandoc's markdown syntax includes extensions for LaTeX math, tables, definition lists, footnotes, and more.

Since the last report, many new features have been added and improvements made. Some highlights:

◦ Support for Textile input and output.
◦ Support for Emacs org-mode output.
◦ A new "builder" module for constructing Pandoc documents programatically.
◦ Support for LaTeXmath macros in markdown documents.
◦ Support for automatic citations and bibliographies using Andrea Rossato's citeproc-hs library.

These last two changes bring two of the most powerful features of LaTeXto pandoc.

Further reading

http://johnmacfarlane.net/pandoc/

8.2.3 DSH — Database Supported Haskell

Report by:	Torsten Grust
Participants:	George Giorgidze, Tom Schreiber, Jeroen Weijers, Alexander Ulrich
Status:	active development

DSH :: → [SQL]

Database-Supported Haskell, DSH for short, is a Haskell library for database-supported program execution. Using the DSH library, a relational database management system (RDBMS) can be used as a coprocessor for the Haskell programming language, especially for those program fragments that carry out data-intensive and data-parallel computations. Rather than embedding a relational language into Haskell, DSH turns idiomatic Haskell programs into SQL queries. The DSH library and the FerryCore package it uses are available on Hackage (http://hackage.haskell.org/package/DSH).

DSH in the Real World. We have used DSH for large scale data analysis. Specifically, in collaboration with researchers working in social and economic sciences, we used DSH to analyse the entire history of Wikipedia (terabytes of data) and a number of online forum discussions (gigabytes of data).

Because of the scale of the data, it would be unthinkable to conduct the data analysis in Haskell without using the database-supported program execution technology featured in DSH. We have formulated several DSH queries directly in SQL as well and found that the equivalent DSH queries were much more concise, easier to write and maintain (mostly due to DSH's support for nesting, Haskell's abstraction facilities and the monad comprehension notation, see below).

One long-term goal is to allow researchers who are not necessarily expert programmers or database engineers to conduct large scale data analysis themselves.

Towards a New Compilation Strategy. As of today, DSH relies on a query compilation strategy coined *loop-lifting*. Loop-lifting comes with important and desirable properties (*e.g.*, the number of SQL queries issued for a given DSH program only depends on the *static type* of the program's result). The strategy, however, relies on a rather complex and monolithic mapping of programs to the relational algebra. To remedy this, we are currently exploring a new strategy based on the *flattening transformation* as conceived by Guy Blelloch. Originally designed to implement the data-parallel declarative language NESL, we revisit flattening in the context of query compilation (which targets database kernels, one particular kind of data-parallel execution environment). Initial results are promising and DSH might switch over in the not too far future. We hope to further improve query quality and also address the formal correctness of DSH's program-to-queries mapping.

Related Work. Motivated by DSH we reintroduced the *monad comprehension* notation into GHC and also extended it for parallel and SQL-like comprehensions. The extension is available in GHC 7.2.

Further reading

http://db.inf.uni-tuebingen.de/research/dsh

8.3 Functional Reactive Programming

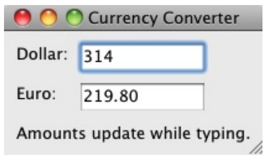
8.3.1 reactive-banana

Report by:	Heinrich Apfelmus
Status:	active development

Reactive-banana is a library for functional reactive programming (FRP). The goal is to create a solid foundation for anything FRP-related.

- Users can finally start experimenting with *graphical user interfaces* based on FRP as the library can be hooked into any existing event-based framework like wxHaskell or Gtk2Hs. A plethora of example code helps with getting started.
- Programmers interested in implementing FRP will have a reference for a *simple semantics* with a working implementation.
- It features an *efficient implementation.* No more spooky time leaks, predicting space & time usage should be straightforward.

Version 0.4.3 of the reactive-banana library has been released on Hackage. It provides a solid push-based implementation of a subset of the semantics for FRP pioneered by Conal Elliott. Compared to the previous report, interoperability with external event frameworks has been improved. The library now also provides many examples, as shown in the screenshot.



Current development focuses on a more integrated notion of time and dynamic event switching.

Further reading

- Project homepage: http://haskell.org/haskellwiki/Reactive-banana
- Example code: http://haskell.org/haskellwiki/Reactive-banana/Examples
- Cabal package: http://hackage.haskell.org/package/reactive-banana
- Developer blog: http://apfelmus.nfshost.com/blog.html

8.3.2 Functional Hybrid Modelling

Report by:	George Giorgidze
Participants:	Joey Capper, Henrik Nilsson
Status:	active research and development

The goal of the FHM project is to gain a better foundational understanding of noncausal, hybrid modelling and simulation languages for physical systems and ultimately to improve on their capabilities. At present, our central research vehicle to this end is the design and implementation of a new such language centred around a small set of core notions that capture the essence of the domain.

Causal modelling languages are closely related to synchronous data-flow languages. They model system behaviour using ordinary differential equations (ODEs) in explicit form. That is, cause-effect relationship between variables must be explicitly specified by the modeller. In contrast, noncausal languages model system behaviour using differential algebraic equations (DAEs) in implicit form, without specifying their causality. Inferring causality from usage context for simulation purposes is left to the compiler. The fact that the causality can be left implicit makes modelling in a noncausal language more declarative (the focus is on expressing the equations in a natural way, not on how to express them to enable simulation) and also makes the models much more reusable.

FHM is an approach to modelling which combines functional programming and noncausal modelling. In particular, the FHM approach proposes modelling with first class models (defined by continuous DAEs) using combinators for their composition and discrete switching. The discrete switching combinators enable modelling of hybrid systems (i.e., systems that exhibit both continuous and discrete dynamic behaviour). The key concepts of FHM originate from work on Functional Reactive Programming (FRP).

We are implementing Hydra, an FHM language, as a domain-specific language embedded in Haskell. The method of embedding employs quasiquoting and enables modellers to use the domain specific syntax in their models. The present prototype implementation of Hydra enables modelling with first class models and supports combinators for their composition and discrete switching.

We implemented support for dynamic switching among models that are computed at the point when they are being “switched in”. Models that are computed at run-time are just-in-time (JIT) compiled to efficient machine code. This allows efficient simulation of structurally dynamic systems where the number of structural configurations is large, unbounded or impossible to determine in advance. This goes beyond to what current state-of-the-art noncausal modelling languages can model. The implementation techniques that we developed should benefit other modelling and simulation languages as well.

We are also exploring ways of utilising the type system to provide stronger correctness guarantees and to provide more compile time reassurances that our system of equations is not unsolvable. Properties such as equational balance (ensuring that the number of equations and unknowns are balance) and ensuring the solvability of locally scoped variables are among our goals. Furthermore, a small core language for FHM is being developed and formalised in the dependently-typed language Agda, allowing us to prove important properties, such as the termination and productivity of the structural dynamics.

In July, this year, I submitted my PhD thesis featuring an in-depth description of the design, semantics and implementation of the Hydra language. In addition, the thesis features a range of example physical systems modelled in Hydra. The examples are carefully chosen to showcase those language features of Hydra that are lacking in other noncausal modelling languages. The next release of Hydra is planned for December, this year. The release will feature all examples from the thesis.

Further reading

The implementation of Hydra and related papers (including my PhD thesis) are available from http://db.inf.uni-tuebingen.de/team/giorgidze.

8.3.3 Elerea

Report by:	Patai Gergely
Status:	experimental, active

Elerea (Eventless reactivity) is a tiny discrete time FRP implementation without the notion of event-based switching and sampling, with first-class signals (time-varying values). Reactivity is provided through various higher-order constructs that also allow the user to work with arbitrary time-varying structures containing live signals.

Stateful signals can be safely generated at any time through a specialised monad, while stateless combinators can be used in a purely applicative style. Elerea signals can be defined recursively, and external input is trivial to attach. The library comes in three major variants, which all have precise denotational semantics:

- `Simple`: signals are plain discrete streams isomorphic to functions over natural numbers;
- `Param`: adds a globally accessible input signal for convenience;
- `Clocked`: adds the ability to freeze whole subnetworks at will.

The code is readily available via cabal-install in the `elerea` package. You are advised to install `elerea-examples` as well to get an idea how to build non-trivial systems with it. The examples are separated in order to minimize the dependencies of the core library. The experimental branch is showcased by Dungeons of Wor, found in the `dow` package (http://www.haskell.org/communities/05-2010/html/report.html#sect6.11.2). Additionally, the basic idea behind the experimental branch is laid out in the WFLP 2010 article *Efficient and Compositional Higher-Order Streams*.

Since the last report, the `Clocked` variant of the library was completely reimplemented. As opposed to the previous version, the new implementation correctly executes according to the desired semantics, and it is also more efficient.

Further reading

- http://hackage.haskell.org/package/elerea
- http://hackage.haskell.org/package/elerea-examples
- http://hackage.haskell.org/package/dow
- http://sgate.emt.bme.hu/documents/patai/publications/PataiWFLP2010.pdf
- http://babel.ls.fi.upm.es/events/wflp2010/video/video-08.html (WFLP talk)

8.4 Audio and Graphics

8.4.1 Audio Signal Processing

Report by:	Henning Thielemann
Status:	experimental, active development

In this project, audio signal algorithms are written in Haskell, that is, no binding to existing sound synthesis systems like SuperCollider. The highlights are:

- It is based on the Numeric Prelude framework (http://haskell.org/communities/05-2009/html/report.html#sect5.6.2).
- We support physical units while maintaining efficiency,
- There are frameworks for abstraction from sample rate. That is, the sampling rate can be omitted in most parts of a signal processing expression.
- We checked several low-level implementations in order to achieve reasonable speed. We complement the standard list type with a lazy `StorableVector` structure and a `StateT s Maybe a` generator, like in stream-fusion. Now, both our custom signal generator type and the Stream type from stream-fusion can be fused to work directly on storable vectors.
- There is support for causal processes. Causal signal processes only depend on current and past data and thus are suitable for real-time processing (in contrast to a function like time reversal). These processes are modeled as `mapAccumL` like functions. Many important operations like function composition maintain the causality property. They are important for sharing on a per sample basis and in feedback loops where they statically warrant that no future data is accessed.
- Type class framework for unifying lazy time values and signals expressed by lists, storable vectors or signal generators.

◦ Connection to ALSA bindings, in order to provide real-time sound synthesis controlled by MIDI events from keyboards or sequencers.
◦ A real-time software synthesizer that employs Just-In-Time-compilation and vector instructions provided by the Low-Level Virtual Machine (http://llvm.org/)

Recent advances are:

◦ Allow to describe acyclic arrow networks using a functional notation. By observation of sharing and the new Vault data structure we get the same efficiency as when using arrow combinators or the arrow syntax.

Further reading

◦ http://www.haskell.org/haskellwiki/Synthesizer
◦ http://hackage.haskell.org/package/vault

8.4.2 Tidal, Texture and Live Music with Haskell

Report by:	Alex McLean
Status:	experimental

For a number of years, I have been improvising live music with Haskell. I have made a pattern library called Tidal and have most recently been working on an experimental visual language on front of that called Texture (formerly known as Text, and I am still in the process of renaming it). There are various videos and some more information on my homepage.

I have been using Tidal and its predecessors in live performance for some years, as shown this video of a performance in Norway: http://piksel.blip.tv/file/4521577/. The quality of the recording is not perfect, but it does show people dancing to Haskell. This performance was with Dave Griffiths (who used his own visual Scheme language SchemeBricks), we perform together (usually as a trio with Adrian Ward) as Slub, and are available for bookings.

Texture is rather experimental, but I recently ran a workshop with it, and got six non-programmers writing Haskell code to improvised music of the acid techno genre together over a few hours.

The code is available at http://darcs.slab.org/, but is undocumented and difficult to get running. Those interested in dabbling in this area would probably be better off looking at hsc3 or haskore. Conductive is another new and interesting library.

At the moment I am finishing off my PhD thesis on a related topic, after that I intend to spend some time packaging Tidal and Texture properly.

Folks interested in Haskell and music, as well as other artforms should consider signing up to the haskell art mailing list.

Further reading

http://yaxu.org/

8.4.3 Hemkay

Report by:	Patai Gergely
Status:	experimental, active

Hemkay (An M.K. Player Whose Name Starts with an H) is a simple music module player that performs all the mixing in Haskell. It supports the popular ProTracker format and some of its variations with different numbers of channels. The device independent mixing functionality can be found in the `hemkay-core` package.

The current version of the player uses the list-based PortAudio bindings for playback, which is highly inefficient.

Since the last update, the mixer went through some performance optimisations. However, the improved mixing performance can only be exploited either through the alternative callback interface of PortAudio (check the `hemkay/callback` branch on GitHub), or through the OpenAL version (`hemkay/openal` branch). Out of the two, the PortAudio version is significantly more efficient, but it is prone to random crashes. Note that this alternative PortAudio binding is only available on GitHub.

Further reading

◦ http://hackage.haskell.org/package/hemkay-core
◦ http://hackage.haskell.org/package/hemkay
◦ http://en.wikipedia.org/wiki/MOD_(file_format)
◦ https://github.com/cobbpg/hemkay
◦ https://github.com/mietek/portaudio

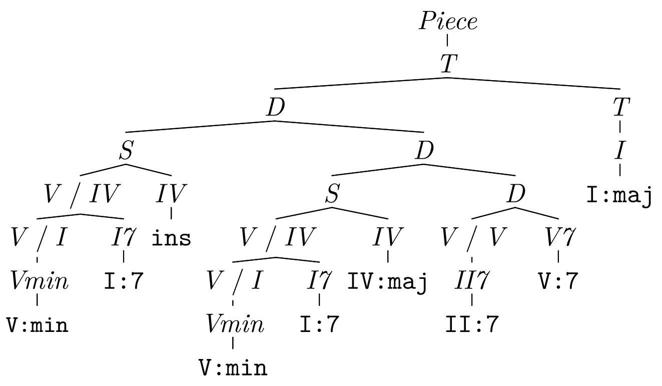
8.4.4 Functional Modelling of Musical Harmony

Report by:	José Pedro Magalhães
Participants:	W. Bas de Haas
Status:	actively developed

Music theory has been essential in composing and performing music for centuries. Within Western tonal music, from the early Baroque on to modern-day jazz and pop music, the function of chords within a chord sequence can be explained by harmony theory. Although Western tonal harmony theory is a thoroughly studied area, formalising this theory is a hard problem.

We have developed a system, named HarmTrace, that formalises the rules of tonal harmony as a Haskell (generalized) algebraic datatype. Given a sequence of chord labels, the harmonic function of a chord in its tonal context is automatically derived. For this, we use several advanced functional programming techniques, such as type-level computations, datatype-generic programming, and error-correcting parsers. We have an experience report at ICFP'11 detailing this project.

As an example, we show a tree representation of the harmony analysis of a short music fragment:



This tree is a visual representation of a value of a Haskell datatype encoding musical harmony, with common notions such as tonic, dominant, etc. Such trees are generated from input sequences of chord labels such as `C:maj F:maj G:7 C:Maj`.

A functional model of harmony offers various benefits: for instance, it can help musicologists in batch-analysing large corpora of digitised scores, but it has proven to be especially useful for solving Music Information Retrieval (MIR) problems. MIR is the research field that aims to provide methods that keep large collections of digital music accessible and maintainable. Hence, besides generating musically meaningful harmonic analyses, HarmTrace explores ways of exploiting these generated analyses to improve the similarity assessment of chord sequences and the automatic extraction for chord labels from musical audio. Some empirical evidence showing that the harmonic analyses of HarmTrace improve harmonic similarity estimation has been published at the International Society for Music Information Retrieval conference 2011.

The code is also available on Hackage.

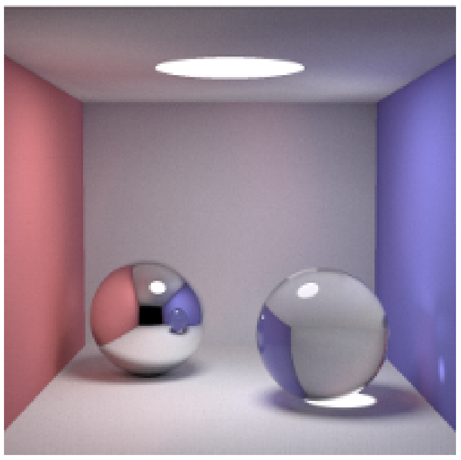
Further reading

http://www.cs.uu.nl/wiki/GenericProgramming/HarmTrace

8.4.5 Cologne

Report by:	Joel Burget
Status:	actively developed

Cologne is a ray tracer being developed in Haskell. The goal is to produce a fun and relatively performant ray tracer. The project has been slowed down recently as my main focus has been on importing more complex models through the Assimp project (→ 7.7.1), but development should pick up this summer. Check out this render of the smallpt scene:



Further reading

https://github.com/joelburget/Cologne

8.5 Hardware Design

8.5.1 CλaSH

Report by:	Christiaan Baaij
Participants:	Arjan Boeijink, Jan Kuper, Anja Niedermeier, Matthijs Kooijman, Marco Gerards
Status:	experimental

CλaSH (CAES Language for Synchronous Hardware) is a functional hardware description language that borrows both its syntax and semantics from Haskell. The clock is implicit for the descriptions made in CλaSH: the behaviour of the circuit is described as transition from the current state to the next, which occurs every clock cycle. The current state is an input of such a transition function, and the updated state part of its result tuple. As descriptions are also valid Haskell, simulations can simply be performed by a Haskell compiler/interpreter (GHC only, due to the use of type families).

Instead of being an embedded language such as ForSyDe (http://www.haskell.org/communities/05-2010/html/report.html#sect6.8.1) and Lava (→ 8.5.2)(→ 10.10), CλaSH has a compiler which can translate Haskell to synthesizable VHDL. The compiler has support for, amongst others: polymorphism, higher-order functions, user-defined algebraic datatypes, and all of Haskell's choice mechanisms. The CλaSH compiler uses GHC for parsing, de-sugaring, and type-checking. The resulting Core-language description is then transformed into a normal form, from which a translation to VHDL is direct. The transformation system uses a set of rewrite rules which are exhaustively applied until a description is in normal form. Examples of these rewrite rules are β-reduction and η-expansion, but also transformations to transform higher-order functions to first-order functions, and transformation for the specialization of polymorphic functions.

The CλaSH compiler was first presented to the community, after 7 months of work, at the Haskell 2009 symposium in Edinburgh, Scotland. Support for arrows and the corresponding syntax, which eases the composition of transition functions, was added in July 2010 and was subsequently presented at IFL 2010 in Alphen a/d Rijn, The Netherlands.

The CλaSH compiler, available as a library, can be found both on Hackage (http://hackage.haskell.org/package/clash, stable) and github (http://github.com/christiaanb/clash/, development). The compiler/interpreter is also available as an executable, which is basically the GHC binary extended with the CλaSH library, on the CλaSH website (http://clash.ewi.utwente.nl).

What is new?

There is now simulation and synthesis support for hardware descriptions that have multiple clock domains, starting with version 0.1.3.0 of CλaSH. Example usage of multiple clock domains is explained here: http://www.haskell.org/pipermail/haskell-cafe/2011-March/090471.html. The code for the demo (which uses multiple clock domains) we did at the DATE'11 conference is available here: http://github.com/christiaanb/DE1-Cyclone-II-FPGA-Board-Support-Package.

Further reading

http://clash.ewi.utwente.nl

8.5.2 Kansas Lava

Report by:	Andy Gill
Participants:	Andy Gill, Andrew Farmer, Ed Komp, Bowe Neuenschwander, Garrin Kimmell (University of Iowa)
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. Kansas Lava draws considerably from Xilinx Lava (http://www.haskell.org/communities/11-2010/html/report.html#sect3.7) and Chalmers Lava (→ 10.10).

The release of Kansas Lava, version 0.2.4, happened in early November. Based round this release, there are a number of resources for users, including a (draft) tutorial, and a youtube channel with walkthroughs of our Lava in use.

On top of Kansas Lava, we are developing Kansas Lava Cores, which was released on hackage at the same time as Kansas Lava. In hardware, a core is a component that can be realized as a circuit, typically on an FPGA. Kansas Lava Cores contains about a dozen cores, and basic board support for Spartan3e, as well as an emulator for the Spartan3e.

Using various components provided as Kansas Lava Cores, we are developing the λ-bridge (→ 8.8.2), with implementations in Haskell and Kansas Lava of a simple protocol stack for communicating with FPGAs. We have early prototypes working, and implementation in Kansas Lava continues.

Finally, we are working on a Lava idiom called a `Patch`, which is a Kansas Lava component interface that uses types to declare protocols and handshakes needed and used. Most of components in the Kansas Lava Cores are instances of our `Patch` idiom. There is a PADL 2012 paper describing `Patch`, including the design and implementation of a controller for an ST7066U-powered LCD display.

Tristan Bull graduated in May 2011 with an MS. His MS thesis was about using Kansas Lava. Congratulations Tristan!

Further reading

- http://www.ittc.ku.edu/csdl/fpg/Tools/KansasLava
- http://www.youtube.com/playlist?list=PL211F8711E3B3DF9C

8.6 Proof Assistants and Reasoning

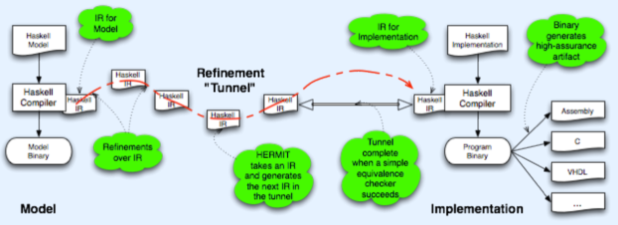
8.6.1 HERMIT

Report by:	Andy Gill
Participants:	Andy Gill, Andrew Farmer, Ed Komp, Nathan Schwermann, PostDoc (TBA)
Status:	active

The Haskell Equational Reasoning Model-to-Implementation Tunnel (HERMIT) is an NSF-funded project being run at KU (→ 10.11) to improving the Applicability of Haskell-Hosted Semi-Formal Models to High Assurance Development. Specifically, HERMIT will use the worker/wrapper transformation, a Haskell-hosted DSL, and a new refinement UI to perform rewrites directly on Haskell Core, the GHC internal representation.

This project is a substantial case study into 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 management scaling issues, data representation and presentation challenges, and understanding the theoretical boundaries of the worker/wrapper transformation.

The project is currently being staffed up, and is expected to run through 2013, and will be released open-source in due time.



Further reading

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

8.6.2 Automated Termination Analyzer for Haskell

Report by:	Jürgen Giesl
Participants:	Matthias Raffelsieper, Peter Schneider-Kamp, Stephan Swiderski, René Thiemann
Status:	actively developed

There are many powerful techniques for automated termination analysis of term rewriting. However, up to now they have hardly been used for real programming languages. We developed an approach which permits the application of existing techniques from term rewriting to prove termination of most functions defined in Haskell programs. In particular, we show how termination techniques for ordinary rewriting can be used to handle those features of Haskell which are missing in term rewriting (e.g., lazy evaluation, polymorphic types, and higher-order functions). We implemented our results in the termination prover AProVE. When testing it on existing standard Haskell-libraries, it turned out that AProVE can fully automatically prove termination of the vast majority of the functions in the libraries.

Further reading

- For details on our approach:

  J. Giesl, M. Raffelsieper, P. Schneider-Kamp, S. Swiderski, and R. Thiemann. Automated Termination Proofs for Haskell by Term Rewriting. *ACM Transactions on Programming Languages and Systems*, 33(2), 2011. http://dx.doi.org/10.1145/1890028.1890030

- To access the implementation via a web interface and for further information on our experiments:

  http://aprove.informatik.rwth-aachen.de/eval/Haskell/

8.6.3 Free Theorems for Haskell

Report by:	Janis Voigtländer
Participants:	Daniel Seidel

Free theorems are statements about program behavior derived from (polymorphic) types. Their origin is the polymorphic lambda-calculus, but they have also been applied to programs in more realistic languages like Haskell. Since there is a semantic gap between the original calculus and modern functional languages, the underlying theory (of relational parametricity) needs to be refined and extended. We aim to provide such new theoretical foundations, as well as to apply the theoretical results to practical problems. The research grant that sponsored Daniel's position has been extended for another round of funding. However, currently we are both consumed by teaching the (by local definition, imperative) programming intro course here at U Bonn, in C (yes, in C), plus an advanced functional programming course, in Haskell.

On the practical side, we maintain a library and tools for generating free theorems from Haskell types, originally implemented by Sascha Böhme and with contributions from Joachim Breitner and now Matthias Bartsch. Both the library and a shell-based tool are available from Hackage (as free-theorems and ftshell, respectively). There is also a web-based tool at http://www-ps.iai.uni-bonn.de/ft/. Features include:

- three different language subsets to choose from
- equational as well as inequational free theorems
- relational free theorems as well as specializations down to function level
- support for algebraic data types, type synonyms and renamings, type classes
- plain text, LaTeX source, PDF, and inline graphics output with nicely typeset theorems

Further reading

http://www.iai.uni-bonn.de/~jv/project/

8.6.4 Streaming Component Combinators

Report by:	Mario Blažević
Status:	experimental, actively developed

See: http://www.haskell.org/communities/11-2010/html/report.html#sect9.6.5.

8.6.5 Swish

Report by:	Douglas Burke
Participants:	Graham Klyne, Vasili I Galchin
Status:	experimental

Swish is a framework for performing deductions in RDF data using a variety of techniques. Swish is conceived as a toolkit for experimenting with RDF inference, and for implementing stand-alone RDF file processors (usable in similar style to CWM, but with a view to being extensible in declarative style through added Haskell function and data value declarations). It explores Haskell as "a scripting language for the Semantic Web", is a work-in-progress, and currently incorporates:

- Support for Turtle, Notation3, and NTriples formats.
- RDF graph isomorphism testing and merging.
- Display of differences between RDF graphs.
- Inference operations in forward chaining, backward chaining and proof-checking modes.
- Simple Horn-style rule implementations, extendable through variable binding modifiers and filters.
- Class restriction rule implementation, primarily for datatype inferences.
- RDF formal semantics entailment rule implementation.
- Complete, ready-to-run, command-line and script-driven programs.

Current Work

A number of incremental changes have been made to the code base, including support for version 7.2 of GHC and some minor optimisations. A parser and formatter for the Turtle format were added, the API changed to use the Text datatype where appropriate, and the vocabulary module was extended to include terms from the Dublin Core, FOAF, Geo and SIOC vocabularies.

Future plans

Continue the clean up and replacement of code with packages from Hacakge. Look for commonalities with the other existing RDF Haskell package, `rdf4h`. Community input — whether it be patches, new code or just feature requests — are more than welcome.

Further reading

- https://bitbucket.org/doug_burke/swish/
- http://www.ninebynine.org/RDFNotes/Swish/Intro.html
- http://protempore.net/rdf4h/

8.7 Natural Language Processing

8.7.1 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. Since the last report, there have been several new releases in the community:

- *brillig* [Hackage]: Partial implementation of the Brill part of speech tagging algorithm (see also the sequor package) (Eric Kow)

- *tokenize* [Hackage]: Simple tokenizer for English text (Grzegorz Chrupala)

- *monad-atom* [Hackage]: Monadically convert objects to unique atoms and back (Grzegorz Chrupala)

- *nlp-scores* [Hackage]: Scoring functions commonly used for evaluation in NLP and IR. Accuracy, Reciprocal Rank, Average Accuracy (Grzegorz Chrupala)

- *Grammatical Framework: Programming with Multilingual Grammars*, Aarne Rante, CSLI Publications, Stanford, 2011, 340 pp, ISBN-10: 1-57586-626-9 (Paper), 1-57586-627-7 (Cloth).

  Also in development are the following packages

- *alpinocorpus-server*: Server for the Alpino treebank library (https://github.com/danieldk/alpinocorpus-server) (Daniel de Kok)

- *alpinocorpus-haskell*: Haskell bindings for the Alpino treebank library. (https://github.com/danieldk/alpinocorpus-haskell) (Daniel de Kok)

At the present, the mailing list is mainly used to make announcements to the Haskell NLP community. Recently, there has been a small uptick in activity, with users looking for NLP libraries on the mailing list. We hope this will further expand and bindings that would be most useful to us and ways of spreading awareness about Haskell in the NLP world.

Further reading

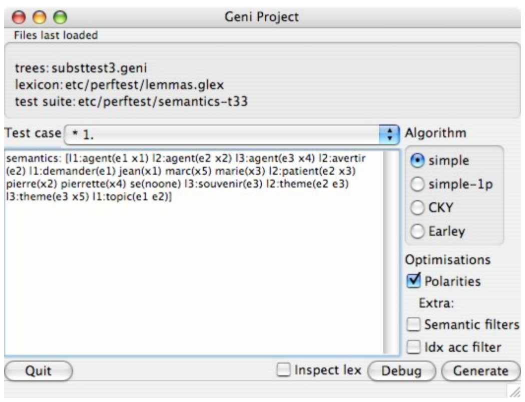
http://projects.haskell.org/nlp

8.7.2 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.

Work on GenI has begun anew. Since May 2011, Eric is working with Computational Linguistics Ltd and SRI international to develop new features for GenI and improve its scalability and performance for use in an interactive tutoring application. We are excited to see GenI potentially being used in the real world!



Files last loaded

trees: substtest3.geni
lexicon: etc/perftest/lemmas.glex
test suite: etc/perftest/semantics-t33

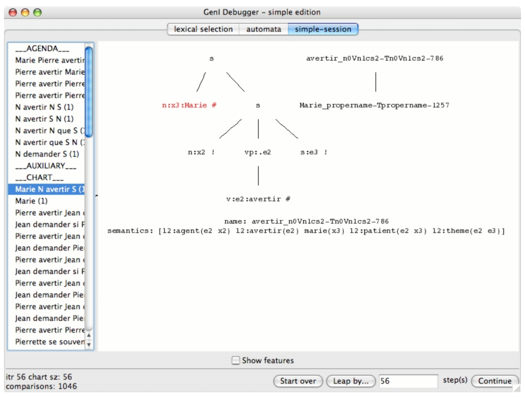
Test case: * 1.

semantics: [l1:agent(e1 x1) l2:agent(e2 x2) l3:agent(e3 x4) l2:avertir (e2) l1:demander(e1) jean(x1) marc(x5) marie(x3) l2:patient(e2 x3) pierre(x2) pierrette(x4) se(noone) l3:souvenir(e3) l2:theme(e2 e3) l3:theme(e3 x5) l1:topic(e1 e2)]

Algorithm: [✓ simple] simple-1p CKY Earley

Optimisations: [✓ Polarities]
Extra: Semantic filters, Idx acc filter

Inspect lex



___AGENDA___
Marie Pierre avertir
Pierre avertir Marie
Pierre avertir Pierre
Pierre avertir Pierre
N avertir N S (1)
N avertir S N (1)
N avertir N que S (
N avertir que S N (
N demander S (1)
___AUXILIARY___
___CHART___
Marie N avertir S (1
Marie (1)
Pierre avertir Jean
Jean demander si P
Pierre avertir Jean
Jean demander Pie
Pierre avertir Jean
Jean demander si P
Pierre avertir Jean
Jean demander Pie
Pierre avertir Jean
Jean demander Pie
Pierre avertir Pierre
Pierrette se souven

name: avertir_n0Vnlcs2-Tn0Vnlcs2-786
semantics: [l2:agent(e2 x2) l2:avertir(e2) marie(x3) l2:patient(e2 x3) l2:theme(e2 e3)]

itr 56 chart sz: 56
comparisons: 1046

56 step(s)

GenI is available on Hackage, and can be installed via cabal-install. Our most recent release of GenI was version 0.20.2 (2009-12-02), with some bugfixes and simplifications. For more information, please contact us on the geni-users mailing list.

Further reading

- 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

8.8 Others

8.8.1 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 developed in co-operation by Ericsson, Chalmers University of Technology (Göteborg, Sweden) and Eötvös Loránd (ELTE) University (Budapest, Hungary).

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 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.

The current version deals with the data-intensive numeric algorithms which are at the core of any DSP application. More recently, we have started to work on extending the language to deal with more system-level aspects such as memory layout and concurrency.

The implementation is available from Hackage.

Further reading

- http://feldspar.inf.elte.hu
- http://hackage.haskell.org/package/feldspar-language
- http://hackage.haskell.org/package/feldspar-compiler

8.8.2 λ-Bridge

Report by:	Andy Gill
Participants:	Andy Gill, Bowe Neuenschwander, Patrick Miller, Ed Komp
Status:	ongoing

The λ-bridge effort provides enabling technology for using functional programming on FPGA fabrics and boards. The majority of the artifacts are shared documentation of ways to use FPGA board, and libraries (software and hardware) that facilitate the use of FPGAs. Techniques for *programming* FPGA boards are well documented, and there are many online examples and other resources to draw from. Getting data to and from a new hardware configuration, however, is a problem every bit (pun intended) as challenging as programming a FPGA in the first place. From empirical evidence, engineers that need communications with a host processor write custom VHDL or Verilog for their specific board to solve this problem. λ-bridge helps solve this problem.

We are using Kansas Lava (→ 8.5.2) to generate various "Cores" that provide a network protocol stack centered round the simple λ-bridge protocol, while being generic about the physical layer. For example, we support the RS-232 cable and UDP over ethernet, and have plans for USB and (where applicable) directly via a motherboard bus. The protocol is also implemented in Haskell, to provided the host-side support. Using the λ-bridge is not the fastest way of communicating with a board, but we hope will be an easy way of getting up and running with a design. Between Kansas Lava, Kansas Lava Cores, and λ-bridge, we plan to introduce a new generation of functional programmers to the joys of FPGA programming.

We are especially interested in using λ-bridge to build a bridge between the statistics language R, and FPGA board, in an attempt to speed up some common statical operations by offshoring the computation to FPGAs.

Further reading

http://www.ittc.ku.edu/csdl/fpg/Tools/LambdaBridge

8.8.3 GenProg — Genetic Programming Library

Report by:	Jan Šnajder
Status:	experimental

The GenProg library is a framework for genetic programming. Genetic programming is an evolutionary technique, inspired by biological evolution, to evolve programs for solving specific problems. A genetic program is represented as an abstract syntax tree and associated with a custom-defined fitness value indicating the quality of the solution. Starting from a randomly generated initial population of genetic programs, the genetic operators of selection, crossover, and (occasionally) mutation are used to evolve programs of increasingly better quality. Standard reference is John Koza's *Genetic programming: On the Programming of Computers by Means of Natural Selection.*

In GenProg, a genetic program is represented by a value of an algebraic datatype. To use a datatype as a genetic program, it suffices to define it as an instance of the `GenProg` typeclass. Any custom datatype can be made an instance of the `GenProg` typeclass. In particular, to use instances of the `Data` typeclass as genetic programs it suffices to define two simple functions: one for the generation of random terminal nodes and another for the generation of random nonterminal nodes. The evolution is governed by several user defined parameters, such as population size, crossover and mutation probabilities, termination criterion, and mutation function. The package is available on Hackage.

Further reading

http://hackage.haskell.org/package/genprog

8.8.4 Manatee

Report by:	Andy Stewart
Status:	active development

Manatee's aim is to build a Haskell Operating System.

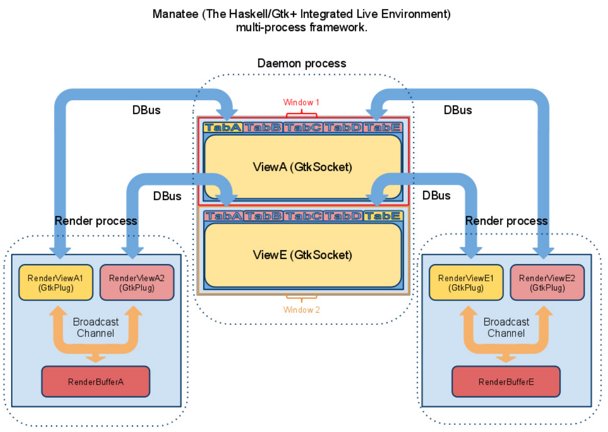
I am an Emacs fan (http://www.emacswiki.org/emacs/AndyStewart) that uses Emacs everyday for everything. But Emacs does not support multi-thread and is not safe enough. So I am building my own Haskell integrated environment — Manatee.

You can write any application in it, and the Manatee framework will mix your application with the current environment. And, most importantly, it gives you a uniform experience with different applications.

Framework

Manatee uses a multi-process framework that makes the extension and the core running in separate processes to protect the application. It will minimize your losses when some unexpected exception happens in the current application; you just need to close/reload the current tab, any other application and the core are still running safely.

Manatee uses a Model-View split design; you can split the current window to get different views for the same buffer (a bit like Emacs's buffers and windows). Then you can mix any applications together with this design for working efficiently.

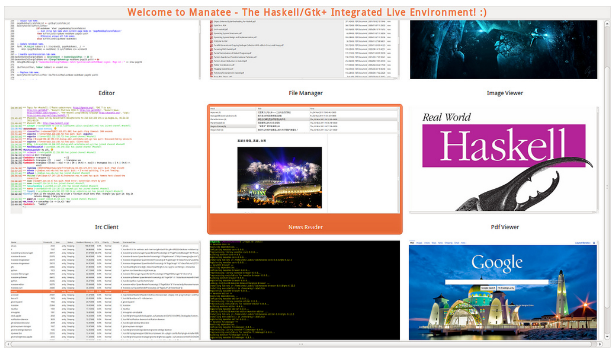


Future plans

I have written the below applications in Manatee:
- Web Browser
- Download Manager
- Editor
- File Manager
- Image Viewer
- IRC Client

- Multimedia Player
- PDF Viewer
- Process Manager
- News Reader
- Terminal

More applications are in development, you are welcome to join us!



Further reading

- Screenshots: http://goo.gl/MkVw
- Videos: http://www.youtube.com/watch?v=weS6zys3U8k, http://www.youtube.com/watch?v=A3DgKDVkyeM
- Wiki page: http://haskell.org/haskellwiki/Manatee

Contact

- Mailing lists: ⟨manatee-user@googlegroups.com⟩, ⟨manatee-develop@googlegroups.com⟩
- IRC channel: irc.freenode.net, 6667, ##manatee

8.8.5 xmonad

Report by:	Gwern Branwen
Status:	active development

XMonad is a tiling window manager for X. Windows are arranged automatically to tile the screen without gaps or overlap, maximizing screen use. Window manager features are accessible from the keyboard; a mouse is optional. XMonad is written, configured, and extensible in Haskell. Custom layout algorithms, key bindings, and other extensions may be written by the user in config files. Layouts are applied dynamically, and different layouts may be used on each workspace. Xinerama is fully supported, allowing windows to be tiled on several physical screens.

Development since the last report has continued; XMonad founder Don Stewart has stepped down and Adam Vogt is the new maintainer. After gestating for 2 years, version 0.10 has been released, with simultaneous releases of the XMonadContrib library of customizations (which has now grown to no less than 216 modules encompassing a dizzying array of features) and the xmonad-extras package of extensions,

Details of changes between releases can be found in the release notes:

- http://haskell.org/haskellwiki/Xmonad/Notable_changes_since_0.8
- http://haskell.org/haskellwiki/Xmonad/Notable_changes_since_0.9
- the Darcs repositories have been upgraded to the hashed format
- XMonad.Config.PlainConfig allows writing configs in a more 'normal' style, and not raw Haskell
- Supports using local modules in xmonad.hs; for example: to use definitions from ˜/.xmonad/lib/XMonad/Stack/MyAdditions.hs
- xmonad –restart CLI option
- xmonad –replace CLI option
- XMonad.Prompt now has customizable keymaps
- Actions.GridSelect - a GUI menu for selecting windows or workspaces & substring search on window names
- Actions.OnScreen
- Extensions now can have state
- Actions.SpawnOn - uses state to spawn applications on the workspace the user was originally on, and not where the user happens to be
- Markdown manpages and not man/troff
- XMonad.Layout.ImageButtonDecoration & XMonad.Util.Image
- XMonad.Layout.Groups
- XMonad.Layout.ZoomRow
- XMonad.Layout.Renamed
- XMonad.Layout.Drawer
- XMonad.Layout.FullScreen
- XMonad.Hooks.ScreenCorners
- XMonad.Actions.DynamicWorkspaceOrder
- XMonad.Actions.WorkspaceNames
- XMonad.Actions.DynamicWorkspaceGroups

Binary packages of XMonad and XMonadContrib are available for all major Linux distributions.

Further reading

- Homepage: http://xmonad.org/
- Darcs source:
  `darcs get` http://code.haskell.org/xmonad
- IRC channel: `#xmonad @@ irc.freenode.org`
- Mailing list: ⟨xmonad@haskell.org⟩

8.8.6 Biohaskell

Report by:	Ketil Malde
Participants:	Christian Höner zu Siederdissen, Nick Ingolia, Felipe Almeida Lessa

Bioinformatics in Haskell is a steadily growing field, and the *Bio* section on Hackage now sports several 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.

Bioinformatics is a diverse field, and consequently, we have different libraries covering mostly separate areas. This summer, some of us participated at the BOSC codefest, and we agreed to factor out common data types that other libraries could use. The result is biocore, currently in revision 0.2. There is an ongoing effort to adapt existing libraries to biocore.

The *biolib library* that supports various sequence and alignment-oriented file formats and operations, is now in the process of being deprecated. Functionality is gradually being factored out, and this has so far resulted in separate libraries for 454 sequencing reads (biosff), and PSL alignment files (biopsl).

The Biobase-prefixed libraries provide basic functionality for a number of data formats. A number of additional libraries are provided: RNAfold is a partial port the ViennaRNA package, MC-Fold-DP: a polynomial-time version of the original MC-Fold pipeline, while RNAwolf provides a novel RNA-folding algorithm with non-canonical secondary structures. On the level of non-coding RNA prediction, CMCompare is used to assess the discriminatory power of RNA family models.

The biostockholm package supports parsing and pretty printing of files in Stockholm 1.0 format. These formats are used by Pfam and Rfam for multiple sequence alignments.

Finally, there is samtools wrapping the samtools C library for accessing and manipulating BAM alignment files, and seqloc providing functionality for manipulating sequence locations and annotation.

Further reading

- http://biohaskell.org
- http://www.tbi.univie.ac.at/~choener/haskell.html

8.8.7 Bullet

Report by:	Csaba Hruska
Status:	experimental, active development

Bullet is a professional open source multi-threaded 3D Collision Detection and Rigid Body Dynamics Library written in C++. It is free for commercial use under the zlib license. The Haskell bindings ship their own (auto-generated) C compatibility layer, so the library can be used without modifications. The Haskell binding provides a low level API to access Bullet C++ class methods. Some bullet classes (Vector, Quaternion, Matrix, Transform) have their own Haskell representation, others are binded as class pointers. The Haskell API provides access to some advanced features, like constraints, vehicle and more.

At the current state of the project most common services are accessible from Haskell, i.e., you can load collision shapes and step the simulation, define constraints, create raycast vehicle, etc. More advanced Bullet features (soft body simulation, Multithread and GPU constaint solver, etc.) will be added later.

Further reading

http://www.haskell.org/haskellwiki/Bullet

8.8.8 Sloth2D

Report by:	Patai Gergely
Status:	experimental, active

Sloth2D is a purely functional 2D physics library with composable high-level abstractions. The primary intent behind this initiative is not to compete with existing engines, but rather to experiment with novel, composable abstractions for physics. This might eventually lead to better high-level interfaces for existing engines, e.g., the Chipmunk and Bullet bindings (→ 8.8.7). However, in the long run it might grow into something that is usable in practice by itself.

The cabalised source is available on GitHub.

Current features:
- 100% pure implementation
- deterministic simulation (replayable regardless of sampling rate)
- convex colliders

Planned features:
- other collider shapes: concave, round, half-plane
- collision layers
- spatial hashing for more efficient collision detection
- object deactivation
- support for raycasting
- serialisation of physics state

- combinators on dynamic worlds
- constraints
- friction
- stacking
- a scene graph-based interface to define the world in a compact manner

Further reading

https://github.com/cobbpg/sloth2d

8.8.9 hledger

Report by:	Simon Michael
Status:	ongoing development; suitable for daily use

hledger is a library and end-user tool (with command-line, curses and web interfaces) for converting, recording, and analyzing financial transactions, using a simple human-editable plain text file format. It is a haskell port and friendly fork of John Wiegley's Ledger, licensed under GNU GPLv3+.

hledger aims to be a reliable, practical tool for daily use. It reports charts of accounts or account balances, filters transactions by type, helps you record new transactions, converts CSV data from your bank, publishes your text journal with a rich web interface, generates simple charts, and provides an API for use in your own financial scripts and apps.

In the last six months there have been two major releases. 0.15 focussed on features and 0.16 focussed on quality. Changes include:

- new modal command-line interface, extensible with hledger-* executables in the path
- more useful web interface, with real account registers and basic charts
- hledger-web no longer needs to create support files, and uses latest yesod & warp
- more ledger compatibility
- misc command enhancements, API improvements, bug fixes, documentation updates
- lines of code increased by 3k to 8k
- project committers increased by 6 to 21

Current plans include:

- Continue the release rhythm of odd-numbered = features, even-numbered = quality/stability/polish, and releasing on the first of a month
- In 0.17, clean up the storage layer, allow rcs integration via filestore, and read (or convert) more formats
- Keep working towards wider usefulness, improving the web interface and providing standard financial reports

Further reading

http://hledger.org

8.8.10 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.

epub-tools is available from Hackage, the Darcs repository below, and also in binary form for Arch Linux through the AUR.

Recent work has been centered on epubname and includes: Smarter parsing of dates in the OPF data, specifically for picking out publication date. The file naming architecture has been completely overhauled, is now more monadic and simpler. It's much easier for a developer to add support for new magazines.

Further reading

- Project page: http://ui3.info/d/proj/epub-tools.html
- Source repository: `darcs get` http://ui3.info/darcs/epub-tools

9 Commercial Users

9.1 Well-Typed LLP

Report by:	Andres Löh
Participants:	Duncan Coutts, Ian Lynagh, Mikolaj Konarski, Nicolas Wu, Eric Kow, Bernie Pope

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⟩.

We are continuing to grow, with currently seven people working as full- or part-time consultants. While we aren't currently officially hiring, we are still regularly looking for fresh blood, so if you would be interested working for us, feel free to check in with us or send us your CV at any time.

We are working for a variety of commercial clients, but naturally, only some of our projects are publically visible.

We continue to be involved in the support of GHC (→3.2). We have contributed to the 7.2.1 release and are currently working towards the upcoming 7.4.1 release.

We coordinate and do work for the Industrial Haskell Group (IHG) (→9.3). We have recently implemented a new dependency solver for Cabal and worked on the Hackage server.

Within the Parallel GHC Project (→5.1.3), we continue to help our old and new partners to implement parallel, concurrent and distributed software in Haskell, and work to improve the tools and libraries, such as ThreadScope.

In addition, we continue to be quite involved in the community, maintaining several packages on Hackage. We have been present at the recent CamHac as well as the Haskell in Leipzig meeting and of course ICFP, Haskell Symposium, Haskell Implementors Workshop and CUFP. We have been teaching at the Utrecht Summer School in Computer Science and the FPDay in Cambridge, and are involved in the Oxford and Munich Haskell user groups.

Several events in the future are currently being planned, we will for example speak at the FP eXchange in London on March 16, 2012, and we will certainly try to participate in the next European Haskell Hackathon.

We are of course always looking for new clients and projects, too, so if you are interested in hiring us, just drop us a mail.

Further reading

- http://www.well-typed.com/
- Blog: http://blog.well-typed.com/

9.2 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

- *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.

9.3 Industrial Haskell Group

Report by:	Andres Löh
Participants:	Duncan Coutts, Ian Lynagh

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.

We welcome two new associate members to the IHG: Silkapp (www.silkapp.com) and Pararallel Scientific.

In the past six months, the collaborative development scheme funded work on cabal-install as well as improvements to the Hackage server. An intermediate status report on the new dependency solver for cabal-install has been presented at the Haskell Implementors Workshop. The solver is available for testing as a branch in the Cabal repository and will soon be merged into the trunk.

Details of the tasks undertaken are appearing on the Well-Typed (→9.1) blog and on the IHG status page.

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.

If you are interested in joining the IHG, or if you just have any comments, please drop us an e-mail at ⟨info@industry.haskell.org⟩.

Further reading

- http://industry.haskell.org/
- http://industry.haskell.org/status/
- http://www.haskell.org/wikiupload/b/b4/HIW2011-Talk-Loeh.pdf
- http://darcs.haskell.org/cabal-branches/cabal-modular-solver/

9.4 Tsuru Capital

Report by:	Bryan Buecking

Tsuru Capital is engaged in high-frequency market-making on options markets. Tsuru is a private company, and trades with its own capital. Tsuru Capital currently runs arbitrage based liquidity provision strategies on the Kospi 200 index and plans to expand to Nikkei 225 index, and other electronic markets, over the next year.

The trading software has been developed entirely in Haskell, and is one of the few systems in the world written completely in a functional language.

Since 2010 we have opened our doors to students, post graduates, and anyone looking for real world experience. And continue to do so by offering paid 3 month internship positions every quarter.

Over the past year we have spent a good deal of time building GUIs for our trading system, and tools for logging and playback. As a result we have contributed bits and pieces of our work to Hackage, and will continue to do so as we flesh out our framework.

Further reading

- http://www.tsurucapital.com/
- http://blog.kfish.org/2011/09/iteratees-at-tsuru.html

9.5 Barclays Capital

Report by: Ben Moseley

Barclays Capital has been using Haskell as the basis for our FPF (Functional Payout Framework) project for about six years now. The project develops a DSL and associated tools for describing and processing exotic equity options.

For the first half of its life the project focused only on the most exotic options — those which the legacy systems were unable 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 and with that the team has grown significantly in both size and geographical distribution. We now have 10 full-time Haskell developers spread between New York, Hong Kong, Kiev and London (with the latter being the biggest development hub).

Our main language is a deeply embedded DSL which has proved very successful, but we are now reaching the stage where some of the traditional DSEL limitations (e.g., error messages and syntactical restrictions) have started to hinder its further adoption. As a result of this we are now working on a new, non-embedded, front-end FPF language which is based on stream arrows and we are investigating the possibility of using the Causal Commutative Arrows approach (Liu, Cheng, Hudak 2009). For the parsing part of this work we have been very impressed by Doaitse Swierstra's uu-parsinglib (→ 7.2.3).

There are a number of other interesting projects going on within the team — these include a new C compiler (compiling our DSL into C) and performance improvement work as this has become more important as our trade population has grown. One interesting aspect of the new C compiler is that it has used an approach inspired by Rodriguez et al's "Generic programming with fixed points for mutually recursive datatypes" to capture the internal AST in a flexible manner through the use of GADTs. (The majority of the rest of our codebase uses standard ADTs but in an unfixed form which facilitates traversals using standard catamorphisms, paramorphisms, apomorphisms etc.).

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.

9.6 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. Currently, Oblomov Systems is working together with Ordina NV on a substantial Haskell project for the Council for the Judiciary in The Netherlands.

Further reading

http://www.oblomov.com

10 Research and User Groups

10.1 A French community for Haskell

Report by:	Alp Mestanogullari
Participants:	Valentin Robert, Fabien Georget, and others
Status:	ongoing

During the past few months, we have seen many new Haskellers in the French-speaking communities. Aside from this, Valentin Robert has published a translation of *Learn You a Haskell For Great Good* in French (see the further reading section). It seems Haskell is finally getting more interest from French developers and that is the reason why we are now trying to create some activity around this.

We are currently working on the basics:

- getting an adequate wiki/website,
- figuring out some project ideas, may they be documentation or software projects,
- working on a first French Hackathon event for French Haskellers to meet and get to know each other.

Among us, we happen to have people interested in many areas (Haskell for web programming, high performance Haskell, etc.) so on the long-term we may be able to provide resources about various topics. Another possible idea would be to have some kind of workgroups that would work on a given project, or even do bug hunting for an already existing project. And we have many other ideas, but it will depend on how the community's activity grows. Our priorities are the website and the first Hackathon, that may happen in June in Strasbourg. This is yet to be confirmed.

We warmly welcome anyone interested in helping us create this community! There are all kinds of tasks to accomplish so you do not need to be a Haskell guru to contribute. We also welcome any French-speaking haskellers or even functional programmers to join us either on the IRC channel `#haskell-fr` on Freenode or on the mailing list.

Further reading

- Homepage: http://www.haskell.fr/
- LYAH in French: http://lyah.haskell.fr/
- Haskell-fr mailing list: http://www.haskell.org/mailman/listinfo/haskell-fr
- Original announcement: http://tinyurl.com/3o9tatf

10.2 Haskell at Eötvös Loránd University (ELTE), Budapest

Report by:	PÁLI Gábor János
Status:	ongoing

Haskell at ELTE

Education

There are many different courses on Haskell and Agda are run at Eötvös Loránd University, Faculty of Informatics.

- Programming for first-year BSc students using Haskell, it is officially in the curriculum.
- 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. ActiveHs is now also avaiable on Hackage.

We started to work on translating our course materials to English, because we are planning to use it for teaching foreign students in the coming years.

This year we organized the Central European Functional Programming Summer School again with more than 60 students. There were many professional lectures delivered by experts on functional programming, like Simon Marlow, Andrew Butterfield, Rinus Plasmeijer, or Mary Sheeran.

Research

We have some research projects in functional programming that use Haskell.

- Feldspar, a high-level domain-specific language for digital signal processing developed for Ericsson in co-operation with Chalmers University of Technology. Our task is to implement an efficient multi-platform ISO C99 code generator for the language.

- Software Technologies for Distributed and Manycore Systems started in 2010. The Project is supported by the European Union and co-financed by the European Social Fund.

Further reading

- Haskell course materials (in English): http://pnyf.inf.elte.hu/fp/Overview_en.xml
- Agda course materials (in English): http://pnyf.inf.elte.hu/fp/Overview_en.xml#agda
- ActiveHs: http://hackage.haskell.org/package/activehs
- CEFP2011: http://plc.inf.elte.hu/cefp/
- Feldspar home page: http://feldspar.inf.elte.hu/

10.3 Functional Programming at UFMG and UFOP

Report by:	Carlos Camarão
Participants:	Marco Gontijo, Lucília Figueiredo, Rodrigo Ribeiro, Cristiano Vasconcellos, Elton Ribeiro
Status:	active development

The Functional Programming groups at Universidade Federal de Minas Gerais and Universidade Federal de Ouro Preto are working on projects that include the following ones:

Proposal for a Solution to Haskell's Multi-parameter Type Class Dilemma The proposal consists of using a simple satisfiability trigger condition: check satisfiability if and only if there exists an unreachable variable in a constraint.

This eliminates the need for functional dependencies (and any other additional mechanism in the language) to tackle ambiguity and overloading resolution.

E-mail messages about the proposal exchanged in Haskell-cafe and Haskell-prime have not been constructive. The discussion in Haskell-cafe deviated to what we see as an ortogonal issue, of import and export of instances (see more about this in the next paragraph).

So unfortunately the proposal has not been incorporated in Haskell yet. A paper about it has been published at SBLP'2009 (see below).

We have implemented the proposal in a proptotype Haskell front-end (https://github.com/rodrigogribeiro/core), and are currently working on this front-end so that it can type all existing Haskell libraries (that use multi-parameter type classes and higher-rank polymorphism).

Controlling the scope of instances in Haskell Marco Gontijo is about to finish his MSc dissertation on the subject. This is a simple and natural change that makes module export and import free of treating instances as a special case. It also allows alternative instances of a class for the same type to be defined and used in different module scopes of a program, eliminates not only problems related to the existence of orphan instances but also the pollution of the global scope by unused instances.

An article about this has been published at SBLP'2011 (see below). Marco Gontijo is currently implementing the proposal, in our Haskell compiler prototype (https://github.com/rodrigogribeiro/core); if time permits, also in GHC.

Decidable type inference for Haskell overloading When types have constraints, decidability of type inference is based mainly on decidability of constraint set satisfiability. We have designed a termination criterion for Haskell's type inference algorithm that deals with all the "complicated cases" (given in e.g. the PPDP'04 and ACM TOPLAS 2005 references below).

A paper about this is being (re)written. An implementation is available at https://github.com/rodrigogribeiro/core.

First Class Overloading and Intersection Types A paper about this has been published at SBLP'2011 (see below).

The work is currently being implemented in our compiler front-end, available at https://github.com/rodrigogribeiro/core.

The Hindley-Milner type system imposes the restriction that function parameters must have monomorphic types. Lifting this restriction and providing system F "first class" polymorphism is clearly desirable, but comes with the difficulty that complete type inference for higher-rank type systems is undecidable. More practical systems supporting higher-rank types have been proposed, which rely on system F, and require appropriate type annotations for the definition of functions with polymorphic type parameters. But these type annotations do inevitably disallow some possible uses of defined higher-rank functions. To avoid this problem, we propose the annotation of intersection types for specifying the types of function parameters used polymorphically inside a function body.

Future work involves extending this work to allow also annotation of union types, supporting then the use (manipulation) of heterogeneous data structures by means of overloaded functions.

Further reading

- *A Solution to Haskell's Multi-paramemeter Type Class Dilemma*, Carlos Camarão, Rodrigo Ribeiro, Lucília Figueiredo, Cristiano Vasconcellos, SBLP'2009 (13th Brazilian Symposium on Programming Languages). http://www.dcc.ufmg.br/~camarao/CT/solution-to-mptc-dilemma.pdf

- *Controlling the Scope of Instances in Haskell*, Marco Silva, Carlos Camarão, SBLP'2011 (15th Brazilian Symposium on Programming Languages). http://www.dcc.ufmg.br/~camarao/controlling-the-scope-of-instances-in-Haskell-sblp2011.pdf
- *Constraint-set satisfiability for Overloading*, Carlos Camarão, Lucília Figueiredo, Cristiano Vasconcellos, ACM Press Conf. Proceedings of PPDP'04 , 67–77, 2004. http://www.dcc.ufmg.br/~camarao/CT/cs-sat/cssat.pdf
- *A theory of overloading*, Peter J. Stuckey, Martin Sulzmann, ACM TOPLAS 2005, 27(6), 1216–1269. http://portal.acm.org/citation.cfm?id=1108974
- *First Class Overloading via Intersection Type Parameters*, Elton Máximo Cardoso, Carlos Camarão, Lucília Figueiredo, SBLP'2011 (15th Brazilian Symposium on Programming Languages). http://www.dcc.ufmg.br/~camarao/CT/intersection-type-parameters.pdf

10.4 Artificial Intelligence and Software Technology at Goethe-University Frankfurt

Report by:	David Sabel
Participants:	Altug Anis, Conrad Rau, Manfred Schmidt-Schauß

Programming language semantics. One of our research topics focuses on programming language semantics, especially on contextual equivalence which is usually based on the operational semantics of the language.

Deterministic call-by-need lambda calculi with letrec provide a semantics for the core language of Haskell. For such an extended lambda calculus we proved correctness of strictness analysis using abstract reduction, and we proved equivalence of the call-by-name and call-by-need semantics. Recently we have shown that applicative bisimilarity is complete w.r.t. contextual equivalence in this calculus.

We also explored several nondeterministic extensions of call-by-need lambda calculi and their applications. A recent result is that for calculi with `letrec` and nondeterminism usual definitions of applicative similarity are unsound w.r.t. contextual equivalence.

We analyzed a higher-order functional language with concurrent threads, monadic IO and synchronizing variables as a core language of Concurrent Haskell. To assure declarativeness of concurrent programming we extended the language by implicit, monadic, and concurrent futures. Using contextual equivalence based on may- and should-convergence, we have shown that various transformations preserve program equivalence, e.g. the monad laws hold in our calculus. Most recently we have shown that the language with concurrency conservatively extends the pure core language of Haskell, i.e. all program equivalences for the pure part also hold in the concurrent language.

In a recent research project we try to automate correctness proofs of program transformations. These proofs require to analyze the overlappings between reductions of the operational semantics and transformation steps by computing so-called forking and commuting diagrams. Recently we implemented an algorithm as a combination of several unification algorithms in Haskell which computes these diagrams. Ongoing research is to automate the corresponding induction proofs (which use the diagrams) using automated termination provers for term rewriting systems.

Grammar based compression. Another research topic of our group focuses on algorithms on grammar compressed strings and trees. One goal is to reconstruct known algorithms on strings and terms (unification, matching, rewriting etc.) for their use on grammars without prior decompression. We recently developed an algorithm for computing the congruence closure on grammar compressed terms. We implemented several algorithms in Haskell which are available as a Cabal package.

Further reading

http://www.ki.informatik.uni-frankfurt.de/research/HCAR.html

10.5 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 — in particular our interest in Erlang has been growing — Haskell provides a major focus and common language for teaching and research.

Our members pursue a variety of Haskell-related projects, some of which are reported in other sections of this report. The third edition of Simon Thompson's text book *Haskell: the craft of functional programming* appeared in June 2011. Thomas Schilling presented his work on improving type error messages for GHC at TFP 2011 and his work on trace-based dynamic optimisations for Haskell programs at IFL 2011. Olaf Chitil is working on more expressive lazy assertions for Haskell.

Further reading

- PLAS group: http://www.cs.kent.ac.uk/research/groups/plas/
- Haskell: the craft of functional programming: http://www.haskellcraft.com
- Refactoring Functional Programs: http://www.cs.kent.ac.uk/research/groups/plas/hare.html
- Tracing and debugging with Hat: http://www.haskell.org/hat
- Heat: http://www.cs.kent.ac.uk/projects/heat/
- Scion: http://code.google.com/p/scion-lib/

10.6 Formal Methods at DFKI and University Bremen

Report by:	Christian Maeder
Participants:	Mihai Codescu, Dominik Dietrich, Christoph Lüth, Till Mossakowski, Lutz Schröder, Ewaryst Schulz
Status:	active development

The activities of our group 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 consists of parsers, static analyzers, and proof tools for languages from 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 ModalCASL. Other languages supported include Haskell (via Programatica), QBF, Maude, VSE, TPTP, THF, OWL, Common Logic, FPL (logic of functional programs) and LF type theory. The Hets implementation is also based on some old Haskell sources such as bindings to uDrawGraph (formerly Davinci) and Tcl/TK that we maintain. Apart from a Gtk2Hs user interface hets also provides many functionalities as a web server based on warp (→ 5.2.2).

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.

The Coalgebraic Logic Satisfiability Solver CoLoSS is being implemented jointly at DFKI Bremen and at the Department of Computing, Imperial College London. The tool is generic over representations of the syntax and semantics of certain modal logics; it uses the Haskell class mechanism, including multi-parameter type classes with functional dependencies, extensively to handle the generic aspects.

Further reading

- Group activities overview:
  http://www.informatik.uni-bremen.de/agbkb/forschung/formal_methods/
- CASL specification language:
  http://www.cofi.info
- Heterogeneous tool set:
  http://www.dfki.de/sks/hets
  http://www.informatik.uni-bremen.de/htk/
  http://www.informatik.uni-bremen.de/uDrawGraph/
- The Coalgebraic Logic Satisfiability Solver CoLoSS:
  http://www.informatik.uni-bremen.de/~lschrode/projects/GenMod
  http://www.doc.ic.ac.uk/~dirk/COLOSS/

10.7 Haskell at Universiteit Gent, Belgium

Report by:	Tom Schrijvers

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:

- As of this academic year, make Haskell part of your curriculum with our brand new *Functional and Logic Programming Languages* course.
- Explore Haskell in depth with one of our Haskell master thesis topics.
- Attend the thriving Ghent Functional Programming Group (→ 10.14).

Research Haskell-related projects of the group members and collaborators are:

- *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.

◦ *Monads, Zippers and Views: Virtualizing the Monad Stack*: We make monadic components more reusable and robust to changes by employing two new techniques for *virtualizing* the monad stack: the *monad zipper* and *monad views*. The monad zipper is a higher-order monad transformer that creates virtual monad stacks by ignoring particular layers in a concrete stack. Monad views provide a general framework for monad stack virtualization: they take the monad zipper one step further and integrate it with a wide range of other virtualizations. For instance, particular views allow restricted access to monads in the stack. Furthermore, monad views provide components with a *call-by-reference*-like mechanism for accessing particular layers of the monad stack. With our two new mechanisms, the monadic effects required by components no longer need to be literally reflected in the concrete monad stack. This makes these components more reusable and robust to changes.

  This is joint work with Bruno Oliveira, part of which is available together with Mauro Jaskelioff's monad transformer library in the Monatron package on Hackage.

◦ *EffectiveAdvice:* EffectiveAdvice is a disciplined model of (AOP-style) advice, inspired by Aldrich's Open Modules, that has full support for effects in both base components and advice. EffectiveAdvice is implemented as a Haskell library. Advice is modeled by mixin inheritance and effects are modeled by monads. Interference patterns previously identified in the literature are expressed as combinators. Equivalence of advice, as well as base components, can be checked by equational reasoning. Parametricity, together with the combinators, is used to prove two harmless advice theorems. The result is an effective model of advice that supports effects in both advice and base components, and allows these effects to be separated with strong non-interference guarantees, or merged as needed. This is joint work with Bruno Oliveira and William Cook.

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/monadiccp

10.8 Haskell in Romania

Report by: Dan Popa

This is to report some activities of the Ro/Haskell Group. The Ro/Haskell page becomes more and more known as time goes. Actually, the Ro/Haskell Group is officially a project of the Faculty of Sciences, "V. Alecsandri" Univ. of Bacău, România (http://stiinte.ub.ro) based by volunteers. During the academic year 2011 – 2012 the "Gentle Introduction to Haskell 98" was translated in Romanian and was published by MatrixRom Publishing House (http://www.matrixrom.ro). Romanian title : "O mica introducere in Haskell 98". Prof Paul Hudak had offered a forward for Romanian users.

Website:

On the 7th of Oct. 2011, the main Ro/Haskell's web page counter recorded the total of almost 40 000 times accessed. Some pages was added, included one dedicate to the above book and some pages dedicated to the Leksah IDE. (http://leksah.org)

Books:

The book "The Practice Of Monadic Interpretation" by Dan Popa had been published in November 2008. The book had developed into a full PhD. thesis which was successfully defended in public in September 2010. No English version is available so far.

Actually the Official Publishing House of the Ro/Haskell Group is MatrixRom (www.matrixrom.ro). Speaking of books, the "Gentle introduction to Haskell" was prepared this year and was on the market in a Romanian translation. The introductory chapter (http://www.haskell.org/wikiupload/3/38/Gentle_1-19-v06-3Aprilie.pdf.zip) can be downloaded from http://www.haskell.org/haskellwiki/Gentle where two other versions are available, too: French and of course English.

"An Introduction to Haskell by Examples" is now out of print but if you need, a special pack can be provided based on the agreement of the author ⟨popavdan@yahoo.com⟩. Also available on special request from PIM Publishing House, in Iasi.

Products:

Haskell products like Rodin (a small DSL a bit like C but written in Romanian) begin to spread, proving the power of the Haskell language. The Pseudocode Language Rodin is used as a tool for teaching basics of Computer Science in some high-schools from various cities. Rodin was asked to become a FOSS (Free & Open Source Software) and will be. To have a sort of C using native keywords was a success in teaching basics of Computer Science: algorithms and structured programming.

Linguists:

A group of researchers from the field of linguistics located at the State Univ. from Bacău (The LOGOS Group) is declaring the intention of bridging the gap between semiotics, high level linguistics, structuralism, nonverbal communication, dance semiotics (and some other intercultural subjects) and Computational Linguistics (meaning Pragmatics, Semantics, Syntax, Lexicology, etc.) using Haskell as a tool for real projects. Probably the situation from Romania is not well known: Romania is probably one of those countries where computational linguistics is studied by computer scientists less than linguists. We had begun by publishing an article about The Rodin Project in order to attract linguists. We are trying to extend the base of available books in libraries.

At Bacău "V. Alecsandri" University

We have teaching Haskell at two Faculties: Sciences (The Computers Science being included) and we hope we will work with Haskell with the TI students from the Fac. of Engineering, where a course on Formal Languages was requested.

At Brasov "Transilvania" University

The book "An Introduction to Haskell by Examples" was requested by teachers from the "Transilvania" Univ. of Brasov., where a master course on functional programming in Haskell was introduced.

Notions:

We are promoting new notions: pseudoconstructors over monadic values (which act both as semantic representations and syntactic structure), modular trees (expanding trees beyound the fixity of the data declarations) and ADFA — adaptive/adaptable determinist finite automata. A dictionary of new notions and concepts is not made, making difficult to launch new ideas and also to track work of the authors.

Unsolved problems:

PhD. advisors (specialized in monads, language engineering, and Haskell) are almost impossible to find. This fact seems to block somehow the hiring of good specialists in Haskell. Also it is difficult to track the Haskell related activity from various universities, like those from: Sibiu, Baia Mare, Timisoara. Please report them using the below address.

Contact

⟨popavdan@yahoo.com⟩

Further reading

- Ro/Haskell: http://www.haskell.org/haskellwiki/Ro/Haskell
- Rodin: http://www.haskell.org/haskellwiki/Rodin
- Gentle introduction to Haskell (Ro): http://www.haskell.org/haskellwiki/Gentle
- ADFA: http://www.haskell.org/haskellwiki/ADFA
- Report from: http://stiinte.ub.ro (the Faculty I belong to)

10.9 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 Thursday of each month. We regularly get 20–30 attendees, with a 70/30 industry/research split. Talks this year have included material on Category Theory, theorem proving, type systems, Template Haskell and a couple of different Haskell libraries. 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/

10.10 Functional Programming at Chalmers

Report by:	Jean-Philippe Bernardy

Functional Programming is an important component of the Department of Computer Science and Engineering at Chalmers. In particular, Haskell has a very important place, as it is used as the vehicle for teaching and numerous projects. Besides functional programming, language technology, and in particular domain specific languages is a common aspect in our projects.

The FP group has two new PostDocs: Moa Johansson and Meng Wang. Moa works on automated reasoning about recursive programs and Meng works on random generation of typed terms.

We have a just started a new 5-year project called "RAW FP: Productivity and Performance through Resource Aware Functional Programming".

Property-based testing QuickCheck is the basis for a European Union project on Property Based Testing (www.protest-project.eu). We are applying the QuickCheck approach to Erlang software, together with Ericsson, Quviq, and others. Much recent work has focused on PULSE, the ProTest User-Level Scheduler for Erlang, which has been used to find race conditions in industrial software — see our ICFP 2009 paper for details. A new tool, QuickSpec, generates algebraic specifications for an API automatically, in the form of equations verified by random testing. We have published about it at TAP 2010; an earlier paper can be found here: http://www.cse.chalmers.se/~nicsma/quickspec.pdf. Lastly, we have devised a technique to speed up testing of polymorphic properties: http://publications.lib.chalmers.se/cpl/record/index.xsql?pubid=99387.

Natural language technology Grammatical Framework (http://www.haskell.org/communities/11-2010/html/report.html#sect9.7.3) 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.

Parser generator and template-haskell BNFC-meta is an embedded parser generator, presented at the Haskell Symposium 2011. Like the BNF Converter, it generates a compiler front end in Haskell. Two things separate BNFC-meta from BNFC and other parser generators:

◦ BNFC-meta is not a program but a library (the parser description is embedded in a quasi-quote).
◦ BNFC-meta automatically provides quasi-quotes for the specified language. This includes a powerful and flexible facility for antiquotation.

More info: http://hackage.haskell.org/package/BNFC-meta.

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 "Proofs for Free" (extensions of the parametricity & dependent types work from ICFP 2010), a Haskell Symposium paper on "Embedded Parser Generators" (see BNFC-meta above) and a workshop on "DSLs for Economical and Environmental Modelling". Patrik Jansson leads a work-package on DSLs within the EU project "Global Systems Dynamics and Policy" (http://www.gsdp.eu/, started Oct. 2010). If you want to apply DSLs, Haskell, and Agda to help modelling global sustainability challenges, please get in touch!

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** (→ 8.8.1) 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 GPGPUs.

The following languages are not actively developed at the moment:

◦ **Lava** is a language for structural hardware description. Circuits are modeled as ordinary Haskell functions, and many of Haskell's advantages (such as higher-order functions and polymorphism) are also available for Lava descriptions. There are several versions of Lava around. The version developed at Chalmers aims particularly at supporting formal verification in a convenient way.

◦ **Wired** is an extension to Lava, targeting (not exclusively) semi-custom VLSI design. A particular aim of Wired is to give the designer more control over on-chip wires' effects on performance. The most recent activity was to use Wired to explore the layout of multipliers (Kasyab P. Subramaniyan, Emil Axelsson, Mary Sheeran and Per Larsson-Edefors. Layout Exploration of Geometrically Accurate Arithmetic Circuits. *Proceedings of IEEE International Conference of Electronics, Circuits and Systems.* 2009).
Home page: http://www.cse.chalmers.se/~emax/wired/.

Automated reasoning 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 analyzing 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.

Teaching Haskell is present in the curriculum as early as the first year of the Bachelors program. We have three courses solely dedicated to functional programming (of which two are Masters-level courses), but we also provide courses which use Haskell for teaching other aspects of computer science, such as programming languages, compiler construction, hardware description and verification, data structures and programming paradigms.

During 2012 the FP group will develop (and teach) a new MSc level course on "Parallel Functional Programming".

10.11 Functional Programming at KU

Report by:	Andy Gill
Status:	ongoing

Functional Programming remains active 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. Apart from Kansas Lava (→ 8.5.2), ChalkBoard (→ 7.7.6), Lambda Bridge (→ 8.8.2) and HERMIT (→ 8.6.1), there are several other FP and Haskell related things going on.

- We are continuing our development of a lightweight Haskell version of HOL. Traditionally, members of the higher-order logic theorem (HOL) proving family have been implemented in the Standard ML programming language or one of its derivatives. HaskHOL aims to break with tradition by implementing a lightweight HOL theorem prover library as a Haskell hosted domain specific language. Based on the HOL Light logical system, HaskHOL aims to provide the ability for Haskell users to reason about their code directly without having to transform it or otherwise export it to an external tool. For details talk to Evan Austin.
- We are developing a library in Haskell for processing Rosetta specifications. A current focus is the modularity and re-use of distinct processing elements, such as type-checking, partial evaluation, and reasoning assistants. Mutually defined elements that are more convenient to consider as distinct interact via a reactive monadic computation, so the two elements' code can be managed as separate packages. Also, our principal specification representation use functors and type-level fixed points to achieve extensibility and generic programming. The goal of the library is to provide to a tight and graduated interface to the basic processing elements, so that the users may incorporate the most appropriate basic elements when implementing their own, more domain-specific Rosetta processors. For details talk to Nick Frisby.

Further reading

- The Functional Programming Group: http://www.ittc.ku.edu/csdl/fpg
- CSDL website: https://wiki.ittc.ku.edu/csdl/Main_Page

10.12 Dutch Haskell User Group

Report by:	Tom Lokhorst

See: http://www.haskell.org/communities/05-2010/html/report.html#sect8.6.

10.13 San Simón Haskell Community

Report by:	Antonio M. Quispe

The San Simón Haskell Community from San Simón University Cochabamba-Bolivia, is an informal Spanish group that aspire to learn, share information, knowledge and experience related to the functional paradigm.

Our main activity is the development of projects, we have some projects in our Web Page (http://comunidadhaskell.org) that serves us as a medium of communication and work environment.

Our last activity was the Local Haskell Hackathon that was held on April 8, 9 and 10 in our University. There were 15 participants of different levels in functional programming. We have been working on projects idbjava (decompiler bytecode java), lexer and parser for Ruby, emulator for CNC machine, and some Haskell games. We have had a wonderful time of 2 days of programming, and I want to thank Vladimir Costas and Pablo Azero for their assistence in the realization of this event.

The next thing we are waiting on is the 2nd Open House Haskell community where we will show some of the projects we are working on.

I want to encourage all Spanish Haskell programmers to meet us on Facebook.

Further reading

http://comunidadhaskell.org

10.14 Ghent Functional Programming Group

Report by:	Jeroen Janssen
Participants:	Bart Coppens, Jasper Van der Jeugt, Tom Schrijvers, Andy Georges, Kenneth Hoste
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.

The first seven GhentFPG meetings and BelHac were reported on in the previous HCARs. Since then we have held two other GhentFPG meetings. GhentFPG #8, held in June 2011, was a combination of talks and a problem-solving activity, where we pickled our brains on a problem from the ACM World Finals Programming Contest. GhentFPG #9 was a regular meeting with one normal talk and three lightning talks:

1. Jurriën Stutterheim — Snaplets: composable and reusable web components.
2. Andy Georges — hCole-server (lightning talk).
3. Kenneth Hoste — GA: a genetic algorithm library (lightning talk).
4. Jasper Van der Jeugt — WebSockets (lightning talk).

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://groups.google.com/group/ghent-fpg