

Haskell Communities and Activities Report

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

Seventeenth Edition — November 2009

Andy Adams-Moran	Janis Voigtländer (ed.)	Krasimir Angelov
Heinrich Apfelmus	Tiago Miguel Laureano Alves	Justin Bailey
Jean-Philippe Bernardy	Dmitry Astapov	Joachim Breitner
Björn Buckwalter	Tobias Bexelius	Roman Cheplyaka
Adam Chlipala	Andrew Butterfield	Jan Christiansen
Alberto Gómez Corona	Olaf Chitil	Jácome Cunha
Nils Anders Danielsson	Duncan Coutts	Facundo Dominguez
Chris Eidhof	Atze Dijkstra	Patai Gergely
Brett G. Giles	Marc Fontaine	George Giorgidze
Dmitry Golubovsky	Andy Gill	Bastiaan Heeren
Claude Heiland-Allen	Jurriaan Hage	Wolfgang Jeltsch
Florian Haftmann	Jan Martin Jansen	Guillaume Hoffmann
Martin Hofmann	Christopher Lane Hinson	Csaba Hruska
Liyang HU	Creighton Hogg	Farid Karimipour
Oleg Kiselyov	Paul Hudak	Michal Konečný
Lyle Kopnicky	Lennart Kolmodin	Bas Lijnse
Ben Lippmeier	Eric Kow	Rita Loogen
Ian Lynagh	Andres Löh	Christian Maeder
José Pedro Magalhães	John MacFarlane	Arie Middelkoop
Ivan Lazar Miljenovic	Michael Marte	Maarten de Mol
Dino Morelli	Neil Mitchell	Rishiyur Nikhil
Thomas van Noort	Matthew Naylor	Miguel Pagano
Jens Petersen	Johan Nordlander	Dan Popa
Jason Reich	Simon Peyton Jones	Alberto Ruiz
David Sabel	Claus Reinke	Uwe Schmidt
Martijn Schrage	Ingo Sander	Paulo Silva
Axel Simon	Tom Schrijvers	Martijn van Steenbergen
Don Stewart	Ganesh Sittampalam	Doaitse Swierstra
Henning Thielemann	Martin Sulzmann	Wren Ng Thornton
Jared Updike	Simon Thompson	Miguel Vilaca
Sebastiaan Visser	Marcos Viera	Kim-Ee Yeoh
	Janis Voigtländer	
	Brent Yorgey	

Preface

This is the 17th 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.

The report is thinner/shorter this time, but has a good percentage of blue and semi-blue entries. I have implemented the strategy, outlined in the May edition, of replacing with online pointers to previous versions those entries for which I received a liveness ping, but which have seen no essential update for a while. 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 around April/May.

Finally, on special occasion, let me pose a prize question. It goes as follows:

How many entries in this report refer to the 2010 PEPM Workshop?

The first correct answer that reaches me wins a free copy of the ACM printed proceedings!

Janis Voigtländer, University of Bonn, Germany, hcar@haskell.org

Contents

1	Information Sources	7
1.1	The Monad.Reader	7
1.2	Haskell Wikibook	7
1.3	Oleg’s Mini tutorials and assorted small projects	7
1.4	Haskell Cheat Sheet	8
1.5	The Happstack Tutorial	8
1.6	Practice of Functional Programming	9
1.7	Cartesian Closed Comic	9
2	Implementations	10
2.1	The Glasgow Haskell Compiler	10
2.2	The Helium compiler	13
2.3	UHC, Utrecht Haskell Compiler	13
2.4	Haskell frontend for the Clean compiler	14
2.5	SAPL, Simple Application Programming Language	14
2.6	The Reduceron	15
2.7	Platforms	15
2.7.1	Haskell in Gentoo Linux	15
2.7.2	Fedora Haskell SIG	15
2.7.3	GHC on OpenSPARC	15
3	Language	17
3.1	Extensions of Haskell	17
3.1.1	Eden	17
3.1.2	XHaskell project	17
3.1.3	HaskellActor	18
3.1.4	HaskellJoin	18
3.2	Related Languages	18
3.2.1	Curry	18
3.2.2	Agda	19
3.2.3	Clean	19
3.2.4	Timber	20
3.2.5	Ur/Web	20
3.3	Type System / Program Analysis	21
3.3.1	Free Theorems for Haskell (and Curry)	21
3.3.2	The Disciplined Disciple Compiler (DDC)	21
4	Tools	22
4.1	Transforming and Generating	22
4.1.1	UUAG	22
4.1.2	AspectAG	22
4.1.3	HFusion	22
4.1.4	Optimus Prime	23
4.1.5	Derive	23
4.1.6	lhs2T _E X	23
4.2	Analysis and Profiling	24
4.2.1	SourceGraph	24
4.2.2	HLint	24
4.2.3	hp2any	24
4.3	Development	25
4.3.1	Hoogle — Haskell API Search	25
4.3.2	HEAT: The Haskell Educational Advancement Tool	25

4.3.3	HaRe — The Haskell Refactorer	25
4.3.4	DarcsWatch	26
4.3.5	HSFFIG	26
5	Libraries	27
5.1	Cabal and Hackage	27
5.2	Haskell Platform	27
5.3	Auxiliary Libraries	28
5.3.1	hmatrix	28
5.3.2	hTensor	28
5.3.3	The Neon Library	28
5.3.4	leapseconds-announced	29
5.4	Parsing and Transforming	29
5.4.1	ChristmasTree	29
5.4.2	Utrecht Parser Combinator Library: New version	29
5.5	Mathematical Objects	29
5.5.1	dimensional: Statically checked physical dimensions	29
5.5.2	Halculon: units and physical constants database	29
5.5.3	Numeric prelude	30
5.5.4	AERN-Real and friends	30
5.5.5	logfloat	31
5.5.6	fad: Forward Automatic Differentiation	31
5.6	Data types and data structures	31
5.6.1	HList — a library for typed heterogeneous collections	31
5.6.2	bytestring-trie	32
5.7	Data processing	32
5.7.1	MultiSetRewrite	32
5.7.2	Graphalyze	32
5.8	Generic and Type-Level Programming	32
5.8.1	uniplate	32
5.8.2	Generic Programming at Utrecht University	32
5.8.3	Extensible and Modular Generics for the Masses (EMGM)	33
5.8.4	Optimizing generic functions	34
5.8.5	2LT: Two-Level Transformation	35
5.8.6	Data.Label — “atoms” for type-level programming	35
5.9	User interfaces	35
5.9.1	Gtk2Hs	35
5.9.2	HQK	36
5.10	Graphics	36
5.10.1	diagrams	36
5.10.2	LambdaCube	36
5.10.3	GPipe	37
5.10.4	ChalkBoard	37
5.10.5	graphviz	38
5.11	Music	38
5.11.1	Haskore revision	38
5.11.2	Euterpea	38
5.12	Web and XML programming	38
5.12.1	Haskell XML Toolbox	38
5.12.2	tagsoup	39
6	Applications and Projects	40
6.1	For the Masses	40
6.1.1	Darcs	40
6.1.2	xmonad	40
6.2	Education	40
6.2.1	Exercise Assistants	40
6.2.2	Holmes, plagiarism detection for Haskell	41

6.2.3	INblobs — Interaction Nets interpreter	41
6.2.4	Yahc	41
6.2.5	grolprep	42
6.3	Web Development	42
6.3.1	Holumbus Search Engine Framework	42
6.3.2	HCluster	43
6.3.3	JavaScript Monadic Writer	43
6.3.4	Haskell DOM Bindings	44
6.3.5	gitit	44
6.4	Data Management and Visualization	45
6.4.1	Pandoc	45
6.4.2	HaExcel — From Spreadsheets to Relational Databases and Back	45
6.4.3	SdfMetz	45
6.4.4	The Proxima 2.0 generic editor	45
6.5	Functional Reactive Programming	46
6.5.1	Functional Hybrid Modelling	46
6.5.2	Elerea	47
6.6	Audio and Graphics	47
6.6.1	Audio signal processing	47
6.6.2	easyVision	48
6.6.3	photoname	48
6.6.4	n-Dimensional Convex Decomposition of Polytopes	48
6.6.5	DVD2473	49
6.6.6	F14m6e	49
6.6.7	GULCI	49
6.6.8	Reflex	50
6.7	Proof Assistants and Reasoning	50
6.7.1	Calculator	50
6.7.2	Saoithín: a 2nd-order proof assistant	50
6.7.3	Inference Services for Hybrid Logics	50
6.7.4	HTab	51
6.7.5	Sparkle	51
6.7.6	Haskabelle	51
6.8	Modeling and Analysis	51
6.8.1	iTasks	51
6.8.2	CSP-M animator and model checker	52
6.9	Hardware Design	52
6.9.1	ForSyDe	52
6.9.2	Kansas Lava	53
6.10	Natural Language Processing	53
6.10.1	NLP	53
6.10.2	GenI	53
6.10.3	Grammatical Framework	54
6.11	Others	55
6.11.1	IgorII	55
6.11.2	Roguestar	55
6.11.3	LQPL — A quantum programming language compiler and emulator	55
6.11.4	Yogurt	56
6.11.5	Dyna 2	56
6.11.6	Vintage BASIC	57
6.11.7	Bullet	57
6.11.8	arbt	57
6.11.9	uacpid	57
7	Commercial Users	58
7.1	Well-Typed LLP	58
7.2	Credit Suisse Global Modeling and Analytics Group	58
7.3	Bluespec tools for design of complex chips	58

7.4	Galois, Inc.	59
7.5	IVU Traffic Technologies AG Rostering Group	60
7.6	Tupil	60
7.7	Aflexi Content Delivery Network (CDN)	60
7.8	Industrial Haskell Group	61
7.9	typLAB	61
8	Research and User Groups	62
8.1	Functional Programming Lab at the University of Nottingham	62
8.2	Artificial Intelligence and Software Technology at Goethe-University Frankfurt	63
8.3	Functional Programming at the University of Kent	63
8.4	Foundations and Methods Group at Trinity College Dublin	64
8.5	Formal Methods at DFKI Bremen and University of Bremen	64
8.6	Haskell at K.U.Leuven, Belgium	64
8.7	Haskell in Romania	65
8.8	fp-syd: Functional Programming in Sydney, Australia.	65
8.9	Functional Programming at Chalmers	66

1 Information Sources

1.1 The Monad.Reader

Report by: Brent Yorgey

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

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

Since the last HCAR there has been one new issue and a change of editors, with Brent Yorgey taking over editorial duties from Wouter Swierstra. The next issue will be published in January.

Further reading

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

1.2 Haskell Wikibook

Report by: Heinrich Apfelmus
Participants: Orzetto, David House, Eric Kow, and other contributors
Status: active development

The goal of the Haskell wikibook project is to build a community textbook about Haskell that is at once free (as in freedom and in beer), gentle, and comprehensive. We think that the many marvelous ideas of lazy functional programming can and thus should be accessible to everyone in a central place. In particular, the wikibook aims to answer all those conceptual questions that are frequently asked on the Haskell mailing lists.

Everyone including you, dear reader, are invited to contribute, be it by spotting mistakes and asking for

clarifications or by ruthlessly rewriting existing material and penning new chapters.

Thanks to the bold action of user Orzetta, the chapter “Understanding monads” finally came to live. It is very useable now, though more work is needed to properly record the current lore about the zoo of standard monads.

Further reading

- <http://en.wikibooks.org/wiki/Haskell>
- Mailing list: wikibook@haskell.org

1.3 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 three additions:

Typed Tagless Interpretations and Typed Compilation

This web page describes embeddings of *typed* domain-specific languages in Haskell, stressing type preservation, typed compilation, and multiple interpretations. Type preservation statically and patently assures that interpreters never get stuck and hence run more efficiently. By ‘typed compilation’ we mean a transformation from an untyped to typed tagless representations. The untyped form is an AST (represented as a regular data type), which is usually the result of parsing a file or similar plain data. The typed tagless representation takes the form of either generalized algebraic data types GADT (the initial approach), or alternatively, type-constructor-polymorphic terms (the final approach). Either type representation can be interpreted in various ways (e.g., evaluated, CPS-transformed, partially evaluated, etc). All these interpretations are assuredly type-preserving and patently free of any ‘type errors’ such as failure to pattern-match type tags or dereferencing an unbound variable.

We show three examples of typed compilation:

Staged Typed Compilation into GADT using typeclasses

A typed compiler is the function of the signature $\text{Expr} \rightarrow \text{Term } t$, where Expr is an ordinary algebraic data type of untyped first-order source terms and $\text{Term } t$ is a GADT. The result *type* t is a function of the *value* of Expr . Thus we demonstrate the Haskell solution of the truly *dependent-type*

problem. We use template Haskell to implement the compiler, so that we get the Haskell type checker itself to type check the embedded DSL.

Typed compilation to tagless-final HOAS The embedded DSL is now higher-order: it is simply-typed lambda-calculus with the fixpoint and constants — essentially, PCF. We no longer use staging. Rather, we compile to tagless-final representation, which can be interpreted by different interpreters. Regardless of the number of interpretations of a term, the type checking happens only once. We build our own Dynamics to encapsulate typed terms and represent their types.

Typed compilation via GADTs This time, we compile the same higher-order DSL to the HOAS tagless-initial representation: GADT. The compiler itself is implemented via GADTs.

The web page also relates Final and Initial typed tagless representations: they are related by bijection.

<http://okmij.org/ftp/Computation/tagless-typed.html>

Delimited continuations with effect typing, full soundness, answer-type modification and polymorphism

We describe the implementations of Asai and Kameyama’s calculus of polymorphic delimited continuations with effect typing, answer-type modification and polymorphism. The calculus has greatly desirable properties of strong soundness (well-typed terms do not give *any* run-time exceptions), principal types, type inference, preservation of types and equality through CPS translation, confluence, and strong normalization for the subcalculus without fix.

Our Haskell 98 code is the first implementation of delimited continuations with answer-type modification, polymorphism, effect typing, and type inference in a widely available language. Thanks to parameterized (generalized) monads the implementation is the straightforward translation of the rules of the calculus. Matthieu Sozeau has defined a generalized monad type-class in the recent version of Coq and so implemented the calculus along with the type-safe `sprintf` in Coq.

<http://okmij.org/ftp/Continuations.html#genuine-shift>

Total stream processors and their applications to all infinite streams

In the article on seemingly impossible functional programs, Martín Escardó wrote about decidable checking of satisfaction of a total computable predicate on Cantor numerals. The latter represent infinite bit strings, or all real numbers within $[0,1]$. Martín Escardó’s technique can tell, *in finite time*, if a given total computable predicate is satisfied over *all* possible infinite

bit strings. Furthermore, for so-called sparse predicates, Martín Escardó’s technique is very fast.

We re-formulate the problem in terms of streams and depth-limited depth-first search, and thus cast off the mystery of deciding the satisfiability of a total computable predicate over the set of all Cantor numerals.

As an additional contribution, we show how to write functions over Cantor numerals in a ‘natural’ monadic style so that those functions become self-partially evaluating. The instantiation of the functions in an appropriate *pure* monad gives us transparent memoization, without any changes to the functions themselves. The monad in question is pure and involves no reference cells. The code is in Haskell 98.

<http://okmij.org/ftp/Algorithms.html#total-sp>

1.4 Haskell Cheat Sheet

Report by:

Justin Bailey

The “Haskell Cheat Sheet” covers the syntax, keywords, and other language elements of Haskell 98. It is intended for beginning to intermediate Haskell programmers and can even serve as a memory aid to experts. The cheat sheet is distributed as a PDF and literate source file. A Spanish translation is also available.

Further reading

<http://cheatsheet.codeslower.com>

1.5 The Happstack Tutorial

Report by:

Creighton Hogg

The Happstack Tutorial aims to be a definitive, up-to-date, resource for how to use the Happstack libraries. I have recently taken over the project from Thomas Hartman. An instance of the Happstack Tutorial is running as a stand-alone website, but in order to truly dig into writing Happstack applications you can cabal install it from Hackage and experiment with it locally.

Happstack Tutorial is updated along with the Happstack Hackage releases, but the darcs head is generally compatible with the darcs head of Happstack.

I am adding a few small tutorials to the package with every release and am always looking for more feedback from beginning Happstack users.

Further reading

- <http://tutorial.happstack.com>
- <http://patch-tag.com/repo/happstack-tutorial>

1.6 Practice of Functional Programming

Report by: Dmitry Astapov
Participants: Lev Walkin, Roman Dushkin, Eugene Kirpichov, Alex Ott, Alex Samoylovich, Kirill Zaborski, Serguey Zefirov, Dmitry Zuiikov
Status: collecting materials for issue #3



“Practice of Functional Programming” is a Russian electronic magazine promoting functional programming, with articles that cover both theoretical and practical aspects of the craft. Most of the material of the already published issues is directly related to Haskell.

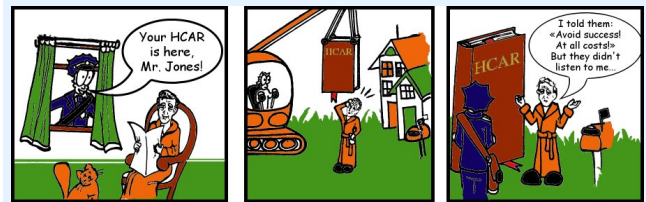
The magazine aims to have a bi-monthly release schedule, with Issue #3 slated for release at the end of November 2009.

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 and #2



Further reading

<http://ro-che.info/ccc/>

1.7 Cartesian Closed Comic

Report by: Roman Cheplyaka
Participants: Maria Kovalyova

Cartesian Closed Comic, or CCC, is a webcomic about Haskell, the Haskell community, and anything else related to Haskell. It is published irregularly. The comic is often inspired by “Quotes of the week” published in Haskell Weekly News. New strips are posted to the Haskell reddit and Planet Haskell. The archives are also available.

Here goes a special edition of the CCC devoted to the Haskell Communities and Activities Report.

2 Implementations

2.1 The Glasgow Haskell Compiler

Report by:	Simon Peyton Jones
Participants:	many others

We are just about to make our annual major release, of GHC 6.12.1 (in the following we will say “GHC 6.12” to refer to GHC 6.12.1 and future patch-level releases along the 6.12 branch).

GHC continues to be very active, with many opportunities for others to get involved. We are particularly eager to find partners who are willing to take responsibility for a particular platform (e.g. Sparc/Solaris, currently maintained by Ben Lippmeier); see <http://hackage.haskell.org/trac/ghc/wiki/Platforms>.

The GHC 6.12 release

We usually try to make a major release of GHC immediately after ICFP. It has been somewhat delayed this year, but we expect to release GHC 6.12 during November or December 2009. Apart from the myriad of new bug fixes and minor enhancements, the big new things in 6.12 are:

- Considerably improved support for parallel execution. GHC 6.10 would execute parallel Haskell programs, but performance was often not very good. Simon Marlow has done lots of performance tuning in 6.12, removing many of the accidental (and largely invisible) gotchas that made parallel programs run slowly.
- As part of this parallel-performance tuning, Satnam Singh and Simon Marlow have developed ThreadScope, a GUI that lets you see what is going on inside your parallel program. It is a huge step forward from “It takes 4 seconds with 1 processor, and 3 seconds with 8 processors; now what?”. ThreadScope will be released separately from GHC, but at more or less the same time as GHC 6.12.
- Dynamic linking is now supported on Linux, and support for other platforms will follow. Thanks for this most recently go to the Industrial Haskell Group (→ 7.8) who pushed it into a fully-working state; dynamic linking is the culmination of the work of several people over recent years. One effect of dynamic linking is that binaries shrink dramatically, because the run-time system and libraries are shared. Perhaps more importantly, it is possible to make dynamic plugins from Haskell code that can be used from other applications.

- The I/O libraries are now Unicode-aware, so your Haskell programs should now handle text files containing non-ascii characters, without special effort.

- The package system has been made more robust, by associating each installed package with a unique identifier based on its exposed ABI. Now, cases where the user re-installs a package without recompiling packages that depend on it will be detected, and the packages with broken dependencies will be disabled. Previously, this would lead to obscure compilation errors, or worse, segfaulting programs.

This change involved a lot of internal restructuring, but it paves the way for future improvements to the way packages are handled. For instance, in the future we expect to track profiled packages independently of non-profiled ones, and we hope to make it possible to upgrade a package in an ABI-compatible way, without recompiling the packages that depend on it. This latter facility will be especially important as we move towards using more shared libraries.

- There are a variety of small language changes, including
 - Some improvements to data types: record punning, declaring constructors with class constraints, GADT syntax for type families etc.
 - You can omit the “\$” in a top-level Template Haskell splice, which makes the TH call look more like an ordinary top-level declaration with a new keyword.
 - We are deprecating mdo for recursive notation, in favour of the more expressive rec statement.
 - We have concluded that the implementation of impredicative polymorphism is unsustainably complicated, so we are re-trenching. It will be deprecated in 6.12 (but will still work), and will be either removed or replaced with something simpler in 6.14.

For more detail, see the release notes in the 6.12 User manual (http://www.haskell.org/ghc/dist/current/docs/html/users_guide/index.html), which mention many things skipped over here.

Internally, GHC 6.12 has a totally re-engineered build system, with much-improved dependency tracking (<http://hackage.haskell.org/trac/ghc/wiki/Building>). While there have been lots of teething problems, things are settling down and the new system is a huge improvement over the old one. The main improvement is that you can usually just say `make`, and everything will be brought up to date (before it

was often necessary to `make clean` first). Another improvement is that the new system exposes much more parallelism in the build, so GHC builds faster on multicores.

GHC and the Haskell platform

Another big change with GHC 6.12 is that Hackage and the Haskell Platform are allowing GHC HQ to get out of the libraries business. So the plan is:

- We release GHC 6.12 with very few libraries.
- Bill Library Author downloads GHC 6.12 and tests his libraries.
- The next Haskell Platform release packages GHC 6.12 with these tested libraries.
- Joe User downloads the Haskell Platform.
- Four months later there is a new HP release, still with GHC 6.12, but with more or better libraries. The HP release cycle is decoupled from GHC.

So if you are Joe User, you want to wait for the HP release. Do not grab the GHC 6.12 release. It will be perfectly usable, but only if you use (an up to date) cabal-install to download libraries, and accept that they may not be tested with GHC 6.12.

What is hot for the next year

GHC continues to be a great substrate for research. Here are the main things we are working on at the moment.

Type systems

Type families have proved a great success. From the outside it might seem that they are done — after all, they are in GHC 6.10 — but the internals are quite fragile and it is amazing that it all works as well as it does. (Thanks to Manuel’s work.) Tom Schrijvers, Dimitrios Vytiniotis, Martin Sulzmann, and Manuel Chakravarty have been working with Simon PJ to understand the fundamentals and, in the light of that insight, to re-engineer the implementation into something more robust. We have developed the “OutsideIn” algorithm, which gives a much nicer account of type inference than our previous story of type inference. The new approach is described in “Complete and Decidable Type Inference for GADTs” [ICFP09a]. More controversially, we now believe that local `let`/where bindings should not be generalised — see “Let should not be generalised” [Let-Gen]. Dimitrios is building a prototype that embodies these ideas, which we will then transfer into GHC.

Meanwhile, Dimitrios, Simon, and Stephanie Weirich are also working on fixing one of GHC’s more embarrassing bugs (<http://hackage.haskell.org/trac/ghc/ticket/1496>), whereby an interaction of type families

and the newtype-deriving can persuade GHC to generate type-unsound code. It has remained un-fixed because the obvious approaches seem to be hacks, so the cure was as bad as the disease. We think we are on to something; stay tuned.

Intermediate language and optimisation

Although it is, by design, invisible to users, GHC’s intermediate language and optimisation passes have been receiving quite a bit of attention. Some highlights:

- Read Max Bolingbroke’s paper on Strict Core [MaxB], a possible new intermediate language for GHC. Adopting Strict Core would be a Big Change, however, and we have not decided to do so (yet).
- Simon PJ totally re-engineered the way that `INLINE` pragmas are implemented, with the goal of making them more robust and predictable (<http://www.haskell.org/pipermail/cvs-ghc/2009-October/050881.html>). There is a new `CONLIKE` pragma which affects rule matching.
- Peter Jonsson did an internship in which he made a start on turning GHC into a supercompiler. Neil Mitchell’s terrific PhD thesis suggested that supercompilation works well for Haskell [NeilM], and Peter has been working on supercompilation for Timber (→ 3.2.4) as part of his own PhD [PeterJ]. The GHC version is not ready for prime time yet, but Simon PJ (now educated by Peter and Neil) is keen to pursue it.
- An internal change in GHC 6.12 is the addition of “annotations”, a general-purpose way for a programmer to add annotations to top-level definitions that can be consulted by a core-to-core pass, and for a core-to-core pass to pass information to its successors (<http://hackage.haskell.org/trac/ghc/wiki/Annotations>). We expect to use these annotations increasingly in GHC itself.

Parallelism

Most of the changes in this area in GHC 6.12.1 were described in our ICFP’09 paper “Runtime Support for Multicore Haskell” [ICFP09b]. The highlights:

- Load-balancing of sparks is now based on lock-free work-stealing queues.
- The overhead for running a spark is significantly less, so GHC can take advantage of finer-grained parallelism
- The parallel GC is now much more locality-aware. We now do parallel GC in young-generation collections by default, mainly to avoid destroying locality by moving data out of the CPU cache on which it is needed. Young-generation collections are parallel

but not load-balanced. There are new RTS flags to control parallel GC behaviour.

- Various other minor performance tweaks.

In the future we plan to focus on the GC, with the main goal being to implement independent per-CPU collection. The other area we plan to look at is changing the GC policy for sparks, as described in our ICFP'09 paper; this will need a corresponding change to the Strategies library to avoid relying on the current “sparks are roots” GC policy, which causes difficulties for writing parallel code that exploits speculation.

Data Parallelism

Data Parallel Haskell has seen few user-visible changes since the last report. Nevertheless, Roman Leshchinskiy has been busy improving many of the fundamental building blocks behind the scenes. These changes were necessary as DPH was able to generate very fast parallel code for simple examples, but the optimisation infrastructure was too fragile — i.e., small changes to other parts of GHC (most notably, the Simplifier) or to the DPH libraries could lead to dramatic performance regressions. Over the last few months, Roman has been working on making the system more robust, while Simon PJ improved and extended parts of GHC's existing optimisation infrastructure (such as the Inliner and other aspects of the Simplifier) to support Roman's efforts. As a first consequence of this recent work, the divide-and-conquer quickhull benchmark (computing a convex hull) is now significantly faster than the corresponding list-based implementation (<http://darcs.haskell.org/packages/dph/examples/quickhull/QuickHullVect.hs>). This is an important milestone as quickhull uses dynamically nested parallelism whose depth is not statically bound.

Gabriele Keller implemented a first prototype of a new library API for regular multi-dimensional arrays to complement the existing irregular, nested arrays. For regular computations on dense matrices, relaxation methods and similar, regular arrays (as opposed to nested arrays) are more convenient and expose additional opportunities for optimisation. Gabriele obtained very encouraging first results with a sequential version that uses a new fusion technique, which we are calling delayed arrays [RegLibBench].

In parallel with the implementation of regular, multi-dimensional arrays as part of DPH, Sean Lee and Manuel Chakravarty are implementing almost the same regular-array API as an EDSL in `Data.Array.Accelerate`. The EDSL implementation restricts the expressiveness of the array language, but at the same time enables us to experiment with more ambitious backends — especially with GPU code generation via CUDA and related technologies. More details are in the video of Manuel's talk from the Haskell Implementors Workshop in Edinburgh [AccelerateTalk].

Code generation

For the last two years we have been advertising a major upheaval in GHC's back end. Currently a monolithic “code generator” converts lambda code (the STG language) into flat C-; “flat” in the sense that the stack is manifested, and there are no function calls. The upheaval splits this into a pipeline of passes, with a relatively-simple conversion of lambda code into C- (with function calls), followed by a succession of passes that optimise this code, and flatten it (by manifesting the stack and removing calls).

John Dias is the principal architect of this new path, and it is in GHC already; you can switch it on by saying `-fnew-codegen`. What remains is (a) to make it work 100% (currently 99%, which is not good enough); (b) commit to it, which will allow us to remove gargantuan quantities of cruft; (c) exploit it, by implementing cool new optimisations at the C- level; (d) take it further by integrating the native code generators into the same pipeline. You can read more on the wiki (<http://hackage.haskell.org/trac/ghc/wiki/Commentary/Compiler/NewCodeGenPipeline>).

Several passes of the new code generation pipeline are supported by Hoopl, a Haskell library that makes it easy to write dataflow analyses and optimisations over C- code [Hoopl]. We think Hoopl is pretty cool, and have well-advanced ideas for how to improve it a lot more.

All of this has taken longer than we hoped. Once the new pipeline is in place we hope that others will join in. For example, David Terei did an interesting undergraduate project on using LLVM as a back end for GHC [Terei], and Krzysztof Wos is just beginning an undergraduate project on optimisation in the new pipeline. We are particularly grateful to Ben Lippmeier for his work on the SPARC native code generator.

Bibliography

ICFP09a Complete and Decidable Type Inference for GADTs. Tom Schrijvers, Simon Peyton Jones, Martin Sulzmann, and Dimitrios Vytiniotis. ICFP'09. <http://research.microsoft.com/~simonpj/papers/gadt>

ICFP09b Runtime Support for Multicore Haskell. Simon Marlow, Satnam Singh, and Simon Peyton Jones. ICFP'09. http://www.haskell.org/~simonmar/bib/multicore-ghc-09_abstract.html

LetGen Let should not be generalized. Dimitrios Vytiniotis, Simon Peyton Jones, and Tom Schrijvers. TLDI'10. <http://research.microsoft.com/~simonpj/papers/constraints/index.htm>

Hoopl Hoopl: dataflow optimisation made simple. Norman Ramsey, John Dias, and Simon Peyton Jones. Rejected by POPL'10. <http://research.microsoft.com/~simonpj/papers/c-->

Terei Low Level Virtual Machine for Glasgow Haskell Compiler. David A. Terei, BSc Thesis. <http://www.cse.unsw.edu.au/~pls/thesis/davidt-thesis.pdf>

MaxB Types are calling conventions. Max Bolingbroke and Simon Peyton Jones. Haskell Symposium 2009. <http://www.cl.cam.ac.uk/~mb566/papers/tacc-hs09.pdf>

NeilM Transformation and Analysis of Functional Programs. Neil Mitchell, PhD thesis, University of York, 2009. <http://community.haskell.org/~ndm/thesis/>

PeterJ Positive supercompilation for a higher order call-by-value language. Peter Jonsson and Johan Nordlander. POPL'09. <http://www.csee.ltu.se/~pj/papers/scp/index.html>

RegLibBench Dense matrix-matrix multiplication benchmark with delayed, regular arrays. <http://www.scribd.com/doc/22091707/Delayed-Regular-Arrays-Sep09>

AccelerateTalk Haskell Array, Accelerated (Using GPUs). Manuel M T Chakravarty, presented at the Haskell Implementors Workshop 2009, Edinburgh. <http://justtesting.posterous.com/running-haskell-array-computations-on-a-gpu>

2.2 The Helium compiler

Report by:	Jurriaan Hage
Participants:	Bastiaan Heeren, Arie Middelkoop

Helium is a compiler that supports a substantial subset of Haskell 98 (but, e.g., $n+k$ patterns are missing). Type classes are restricted to a number of built-in type classes and all instances are derived. The advantage of Helium is that it generates novice friendly error feedback. The latest versions of the Helium compiler are available for download from the new website located at <http://www.cs.uu.nl/wiki/Helium>. This website also explains in detail what Helium is about, what it offers, and what we plan to do in the near and far future.

We are still working on making version 1.7 available, mainly a matter of updating the documentation and testing the system. Internally little has changed, but the interface to the system has been standardized, and the functionality of the interpreters has been improved and made consistent. We have made new options available (such as those that govern where programs are logged to). The use of Helium from the interpreters is now governed by a configuration file, which makes the use of Helium from the interpreters quite transparent for the programmer. It is also possible to use different versions of Helium side by side (motivated by the development of Neon (\rightarrow 5.3.3)).

A student has added parsing and static checking for type class and instance definitions to the language, but

type inferencing and code generating still need to be added. The work on the documentation has progressed quite a bit, but there has been little testing thus far, especially on a platform such as Windows.

2.3 UHC, Utrecht Haskell Compiler

Report by:	Atze Dijkstra
Participants:	Jeroen Fokker, Doaitse Swierstra, Arie Middelkoop, Lucília Camarão de Figueiredo, Carlos Camarão de Figueiredo, Vincent van Oostrum, Clemens Grabmayer, Tom Lokhorst, Jeroen Leeuwestein, Atze van der Ploeg, Paul van der Ende
Status:	active development

UHC, what is new? UHC is the Utrecht Haskell Compiler, supporting almost all Haskell 98 features plus experimental extensions. The first release of UHC was announced on April 18, 2009, at the 5th Haskell Hackathon, held in Utrecht.

Since then we have been working on:

- o A new garbage collector (GC) to replace the Boehm GC we have been using. The new GC is constructed relatively independent of the UHC runtime system, so as to allow multiple backends to plugin backend specific info, e.g., about the memory layout of memory cells. The new GC is used by the bytecode interpreter backend (Atze Dijkstra).
- o Making nofib testsuite examples run. This also has been the driving force and testbed for the new GC (Jeroen Fokker).
- o Typed core, which combines GHC's core language with Henk and recent work on types as calling conventions (Atze Dijkstra).

UHC, what do we plan? Soon we start working on the research grant for "Realising Optimal Sharing", based on work by Vincent van Oostrum and Clemens Grabmayer.

We plan a next release of UHC with the new garbage collector and various bugfixes. We had hoped to offer a complete Haskell 98 library and Cabal support, but likely this will have to wait for a subsequent release.

Furthermore, the following student projects are underway or soon start:

- o Various static analyses on typed core (Tom Lokhorst).
- o Incrementalization of whole program analysis (Jeroen Leeuwestein).
- o Lazy closures (Atze van der Ploeg).
- o GC & LLVM (Paul van der Ende).

Finally, still going on are the following:

- GRIN backend, whole program analysis (Jeroen Fokker).
- Type system formalization and automatic generation from type rules (Lucília Camarão de Figueiredo, Arie Middelkoop).

Background info UHC actually is a series of compilers of which the last is UHC, plus an aspectwise organized infrastructure for facilitating experimentation and extension. The end-user will probably only be aware of UHC as a Haskell compiler, whereas compiler writers will be more aware of the internals of UHC.

For the description of UHC an Attribute Grammar system (→ 4.1.1) is used as well as other formalisms allowing compact notation like parser combinators. For the description of type rules, and the generation of an AG implementation for those type rules, we use the Ruler system. For source code management we use Shuffle, which allows partitioning the system into a sequence of steps and aspects. (Both Ruler and Shuffle are included in UHC).

The implementation of UHC also tackles other issues:

- To deal with the inherent complexity of a compiler, the implementation of UHC is organized as a series of increasingly complex steps. Each step corresponds to a Haskell subset which itself is an extension of the previous step. The first step starts with the essentials, namely typed lambda calculus; the last step corresponds to UHC.
- Independent of each step the implementation is organized into a set of aspects. Currently the type system and code generation are defined as aspects, which can then be left out so the remaining part can be used as a barebones starting point.
- Each combination of step + aspects corresponds to an actual, that is, an executable compiler. Each of these compilers is a compiler in its own right.
- The description of the compiler uses code fragments which are retrieved from the source code of the compilers. In this way the description and source code are kept synchronized.

Part of the description of the series of EH compilers is available as a PhD thesis.

Further reading

- UHC Homepage: <http://www.cs.uu.nl/wiki/UHC/WebHome>
- 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>

2.4 Haskell frontend for the Clean compiler

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

We are currently working on a frontend for the Clean compiler (→ 3.2.3) that supports a subset of Haskell 98. This will allow Clean modules to import Haskell modules, and vice versa. Furthermore, we will be able to use some of Clean's features in Haskell code, and vice versa. For example, we could define a Haskell module which uses Clean's uniqueness typing, or a Clean module which uses Haskell's newtypes. The possibilities are endless!

Future plans

Although a beta version of the new Clean compiler is released early this year to the institution in Nijmegen, there is still a lot of work to do before we are able to release it to the outside world. So we cannot make any promises regarding the release date. Just keep an eye on the Clean mailing lists for any important announcements!

Further reading

http://wiki.clean.cs.ru.nl/Mailing_lists

2.5 SAPL, Simple Application Programming Language

Report by:	Jan Martin Jansen
Status:	experimental, active development

SAPL is an experimental interpreter for a lazy functional intermediate language. The language is more or less equivalent to the core language of Clean (→ 3.2.3). SAPL implementations in C and Java exist. It is possible to write SAPL programs directly, but the preferred use is to generate SAPL. We already implemented an experimental version of the Clean compiler that generates SAPL as well. The Java version of the SAPL interpreter can be loaded as a PlugIn in web applications. Currently we use it to evaluate tasks from the iTask system (→ 6.8.1) at the client side and to handle (mouse) events generated by a drawing canvas PlugIn.

Future plans

For the near future we have planned to make the Clean to SAPL compiler available in the standard Clean distribution. Also some further performance improvements of SAPL are planned.

Further reading

- <http://home.hetnet.nl/~janmartinjansen/saplinter>
- <http://home.hetnet.nl/~janmartinjansen/lambda>
- <http://www.st.cs.ru.nl/Onderzoek/Publicaties/publicaties.html>

2.6 The Reduceron

Report by:	Matthew Naylor
Participants:	Colin Runciman, Jason Reich
Status:	experimental

Over the past year, work on the Reduceron has continued, and we have reached our goal of improving runtime performance by a factor of six! This has been achieved through many small improvements, spanning architectural, runtime, and compiler-level advances.

Two main by-products have emerged from the work. First, *York Lava*, now available from Hackage, is the HDL we use. It is very similar to Chalmers Lava, but supports a greater variety of primitive components, behavioural description, number-parameterised types, and a first attempt at a Lava prelude. Second, *F-lite* is our subset of Haskell, with its own lightweight toolset.

There remain some avenues for exploration. We have taken a step towards parallel reduction in the form of *speculative evaluation of primitive redexes*, but have not yet attempted the *Reducera* — multiple Reducerons running in parallel. And recently, Jason has been continuing his work on the F-lite supercompiler (→ 4.1.4), which is now producing some really nice results.

Alas, the time to take stock and publish a full account of what we have already done is rapidly approaching!

Further reading

- <http://www.cs.york.ac.uk/fp/reduceron/>
- <http://hackage.haskell.org/package/york-lava/>

2.7 Platforms

2.7.1 Haskell in Gentoo Linux

Report by:	Lennart Kolmodin
------------	------------------

Gentoo Linux currently supports GHC 6.10.4, including the latest Haskell Platform (→ 5.2) for x86 and amd64. For previous GHC versions we have binaries available for alpha, amd64, hppa, ia64, sparc, and x86.

Browse the packages in portage 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 a Haskell overlay providing another 300 packages. Thanks to the Haskell developers using Cabal and Hackage (→ 5.1), we have been able to write a tool called “hackport” (initiated by Henning Günther) to generate Gentoo packages that rarely need much tweaking.

Read about the Gentoo Haskell Overlay at <http://haskell.org/haskellwiki/Gentoo>. Using Darcs (→ 6.1.1), it is easy to keep updated and send patches. It is also available via the Gentoo overlay manager “layman”. If you choose to use the overlay, then problems should be reported on IRC (#gentoo-haskell on freenode), where we coordinate development, or via email (haskell@gentoo.org).

Through recent efforts we have developed a tool called “haskell-updater” <http://www.haskell.org/haskellwiki/Gentoo#haskell-updater>. It helps the user when upgrading GHC versions, fixes breakages from library upgrades, etc.

As always we are happy to get help hacking on the Gentoo Haskell framework, hackport, writing ebuilds, and supporting users. Please contact us on IRC or email if you are interested!

2.7.2 Fedora Haskell SIG

Report by:	Jens Petersen
Participants:	Bryan Sullivan, Yaakov Nemoy, Zach Oglesby, Conrad Meyer, Fedora Haskell SIG
Status:	on-going

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

Fedora 12 has just shipped with ghc-6.10.4, Haskell Platform-2009.2.0.2, xmonad, and various new libraries (cgi, editline, fgl, GLUT, network, OpenGL, tar, time, utf8-string, X11-xft, xmonad-contrib).

Fedora 13 is expected to ship with ghc-6.12 with shared libraries, and more new packages in about 6 months.

Contributions to Fedora Haskell are welcome: join us on #fedora-haskell on Freenode IRC.

Further reading

- <http://fedoraproject.org/wiki/SIGs/Haskell>
- http://fedoraproject.org/wiki/Documentation_Development_Tools_Beat#Haskell

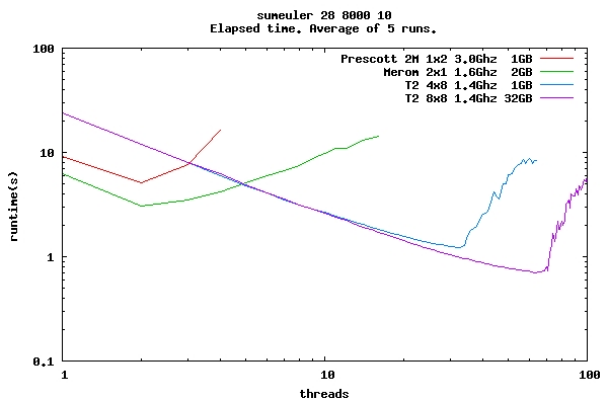
2.7.3 GHC on OpenSPARC

Report by:	Ben Lippmeier
Participants:	Duncan Coutts, Darryl Gove, Roman Leshchinskiy
Status:	winding down

Through January–April this year I repaired GHC’s back end support for the SPARC architecture, and

benchmarked its performance on haskell.org's shiny new SPARC T2 server. I also spent time refactoring GHC's native code generator to make it easier to understand and maintain, and thus less likely for pieces to suffer bit-rot in the future.

The T2 architecture is interesting to functional programmers because of its highly multi-threaded nature. The T2 has eight cores with eight hardware threads each, for a total of 64 threads per processor. When one of the threads suffers a cache miss, another can continue on with little context switching overhead. All threads on a particular core also share the same L1 cache, which supports fast thread synchronization. This is a perfect fit for parallel lazy functional programs, where memory traffic is high, but new threads are only a `par` away. The following graph shows the performance of the `suneuler` benchmark from the `nofib` suite when running on the T2. Note that the performance scales almost linearly (perfectly) right up to the point where it runs out of hardware threads.



The project is nearing completion, pending tying up some loose ends, but the port is fully functional and available in the current head branch. More information, including benchmarking is obtainable from the link below. The GHC on OpenSPARC project was generously funded by Sun Microsystems.

Further reading

<http://ghcsparc.blogspot.com>

3 Language

3.1 Extensions of Haskell

3.1.1 Eden

Report by:	Rita Loogen
Participants:	in Madrid: Ricardo Peña, Yolanda Ortega-Mallén, Mercedes Hidalgo, Fernando Rubio, Alberto de la Encina, Lidia Sánchez-Gil in Marburg: Jost Berthold, Mischa Dieterle, Thomas Horstmeyer, Oleg Lobachev, Rita Loogen
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 function `process :: (a -> b) -> Process a b` embeds a function of type `(a -> b)` into a *process abstraction* of type `Process a b` which, when instantiated, will be executed in parallel. *Process instantiation* is expressed by the predefined infix operator `(#) :: Process a b -> a -> b`. Higher-level coordination is achieved by defining *skeletons*, ranging from a simple parallel map to sophisticated replicated-worker schemes. They have been used to parallelize a set of non-trivial benchmark programs.

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.

Implementation

A major revision of the parallel Eden runtime environment for GHC 6.8.1 is available from the Marburg group on request. Support for Glasgow parallel Haskell (<http://haskell.org/communities/05-2009/html/report.html#sect3.1.2>) is currently being added to this version of the runtime environment. It is planned for the future to maintain a common parallel runtime environment for Eden, GpH, and other parallel Haskell. A parallel Haskell Hackathon will take place in St Andrews from December 10th till 12th, 2009 to join the

various activities and develop the common parallel runtime environment further.

Parallel program executions can be visualized using the Eden trace viewer tool EdenTV. Recent results show that the Eden system behaves equally well on workstation clusters and on multi-core machines.

Recent and Forthcoming Publications

- o Mischa Dieterle, Thomas Horstmeyer, Rita Loogen: *Skeleton Composition Using Remote Data*, in: Practical Aspects of Declarative Programming 2010 (PADL'10), Madrid, Spain, January 2010, Springer LNCS, to appear.
- o Alberto de la Encina, I. Rodríguez, Fernando Rubio: *pHood: A Tool to Analyze Parallel Functional Programs*, in: Symposium on the Implementation and Application of Functional Languages (IFL) 2009, Technical Report, SHU-TR-CS-2009-09-1, Seton Hall University, New York, USA, September 2009, 85–99.
- o Jost Berthold, Mischa Dieterle, and Rita Loogen: *Implementing Parallel Google Map-Reduce in Eden*, in: EuroPar 2009, Delft, NL, Springer LNCS 5704, 990–1002.
- o Jost Berthold, Mischa Dieterle, Oleg Lobachev, Rita Loogen: *Parallel FFT With Eden Skeletons*, in PaCT 2009, Novosibirsk, Russia, Springer LNCS 5698, 73–83.
- o Jost Berthold, Simon Marlow, Abyd Al Zain, and Kevin Hammond: *Comparing and optimising parallel Haskell implementations on multicore*, In Tomoya Enokido et al., editors, 3rd Int. Workshop on Advanced Distributed and Parallel Network Applications (ADPNA-2009) IEEE Computer Society, 2009.
- o Lidia Sánchez-Gil, Mercedes Hidalgo-Herrero, Yolanda Ortega-Mallén: *An Operational Semantics for Distributed Lazy Evaluation*, Trends in Functional Programming, Volume 10, Intellect 2009, to appear.
- o Thomas Horstmeyer, Rita Loogen: *Grace — Graph-based Communication in Eden*, Trends in Functional Programming, Volume 10, Intellect 2009, to appear.

Further reading

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

3.1.2 XHaskell project

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

XHaskell is an extension of Haskell which combines parametric polymorphism, algebraic data types, and type classes with XDuce style regular expression types, subtyping, and regular expression pattern matching. The latest version can be downloaded via <http://code.google.com/p/xhaskell/>

Latest developments

The latest version of the library-based regular expression pattern matching component is available via the google code web site. We are currently working on a paper describing the key ideas of the approach.

3.1.3 HaskellActor

Report by:	Martin Sulzmann
Status:	stable

The focus of the HaskellActor project is on Erlang-style concurrency abstractions. See for details: <http://sulzmann.blogspot.com/2008/10/actors-with-multi-headed-receive.html>.

Novel features of HaskellActor include

- Multi-headed receive clauses, with support for
- guards, and
- propagation

The HaskellActor implementation (as a library extension to Haskell) is available via <http://hackage.haskell.org/cgi-bin/hackage-scripts/package/actor>.

The implementation is stable, but there is plenty of room for optimizations and extensions (e.g. regular expressions in patterns). If this sounds interesting to anybody (students!), please contact me.

Latest developments

We are currently working towards a distributed version of Haskell actor following the approach of Frank Huch, Ulrich Norbistrath: Distributed Programming in Haskell with Ports, IFL'00.

3.1.4 HaskellJoin

Report by:	Martin Sulzmann
Status:	stable

HaskellJoin is a (library) extension of Haskell to support join patterns. Novelties are

- guards and propagation in join patterns,
- efficient parallel execution model which exploits multiple processor cores.

Latest developments

In this honors thesis, Olivier Pernet (a student of Susan Eisenbach) provides a nicer monadic interface to the HaskellJoin library.

Further reading

<http://sulzmann.blogspot.com/2008/12/parallel-join-patterns-with-guards-and.html>

3.2 Related Languages

3.2.1 Curry

Report by:	Jan Christiansen
Participants:	Bernd Braßel, Michael Hanus, Wolfgang Lux, Sebastian Fischer, and others
Status:	active development

Curry is a functional logic programming language with Haskell syntax. In addition to the standard features of functional programming like higher-order functions and lazy evaluation, Curry supports features known from logic programming. This includes programming with non-determinism, free variables, constraints, declarative concurrency, and the search for solutions. Although Haskell and Curry share the same syntax, there is one main difference with respect to how function declarations are interpreted. In Haskell the order in which different rules are given in the source program has an effect on their meaning. In Curry, in contrast, the rules are interpreted as *equations*, and overlapping rules induce a non-deterministic choice and a search over the resulting alternatives. Furthermore, Curry allows to call functions with free variables as arguments so that they are bound to those values that are demanded for evaluation, thus providing for function inversion.

There are three major implementations of Curry. While the original implementation PAKCS (Portland Aachen Kiel Curry System) compiles to Prolog, MCC (Münster Curry Compiler) generates native code via a standard C compiler. The Kiel Curry System (KiCS) compiles Curry to Haskell aiming to provide nearly as good performance for the purely functional part as modern compilers for Haskell do. From these implementations only MCC will provide type classes in the near future. Type classes are not part of the current definition of Curry, though there is no conceptual conflict with the logic extensions.

Recently, new compilation schemes for translating Curry to Haskell have been developed that promise significant speedups compared to both the former KiCS implementation and other existing implementations of Curry.

There have been research activities in the area of functional logic programming languages for more than a decade. Nevertheless, there are still a lot of interesting research topics regarding more efficient compila-

tion techniques and even semantic questions in the area of language extensions like encapsulation and function patterns. Besides activities regarding the language itself, there is also an active development of tools concerning Curry (e.g., the documentation tool Curry-Doc, the analysis environment CurryBrowser, the observation debuggers COOSy and iCODE, the debugger B.I.O. (http://www-ps.informatik.uni-kiel.de/currywiki/tools/oracle_debugger), EasyCheck (<http://haskell.org/communities/05-2009/html/report.html#sect4.3.2>), and CyCoTest). Because Curry has a functional subset, these tools can canonically be transferred to the functional world.

Further reading

- <http://www.curry-language.org/>
- <http://wiki.curry-language.org/>

3.2.2 Agda

Report by:	Nils Anders Danielsson
Participants:	Ulf Norell 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.

New since last time:

- Version 2.2.4 has been released.
- Agda is now available in Ubuntu.
- Darin Morrison is currently extending Cabal to support Agda code.

Further reading

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

3.2.3 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. Clean is the only functional language in the world which offers uniqueness typing. This type system makes it possible in a pure functional language to incorporate destructive updates of arbitrary data structures (including arrays) and to make direct interfaces to the outside imperative world.

Here is a short list with 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.
- The uniqueness type 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. The uniqueness type system can be also 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 non-functional world (to C but also to I/O systems like X-Windows) offering direct access to file systems and operating systems.
- The Clean type system supports dynamic types, 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 supports type classes and type constructor classes to make overloaded use of functions and operators possible.
- Clean offers records and (destructively updateable) arrays and files.
- Clean has pattern matching, guards, list comprehensions, array comprehensions and a lay-out sensitive mode.
- Clean offers a sophisticated I/O library with which window based interactive applications (and the handling of menus, dialogs, windows, mouse, keyboard, timers, and events raised by sub-applications) can

be specified compactly and elegantly on a very high level of abstraction.

- There is a Clean IDE and there are many libraries available offering additional functionality.

Future plans

Please see the entry on a Haskell frontend for the Clean compiler (→ 2.4) for the future plans.

Further reading

- <http://clean.cs.ru.nl/>
- <http://wiki.clean.cs.ru.nl/>

3.2.4 Timber

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

Timber is a general programming language derived from Haskell, with the specific aim of supporting development of complex event-driven systems. It allows programs to be conveniently structured in terms of objects and reactions, and the real-time behavior of reactions can furthermore be precisely controlled via platform-independent timing constraints. This property makes Timber particularly suited to both the specification and the implementation of real-time embedded systems.

Timber shares most of Haskell’s syntax but introduces new primitive constructs for defining classes of reactive objects and their methods. These constructs live in the *Cmd* monad, which is a replacement of Haskell’s top-level monad offering mutable encapsulated state, implicit concurrency with automatic mutual exclusion, synchronous as well as asynchronous communication, and deadline-based scheduling. In addition, the Timber type system supports nominal subtyping between records as well as datatypes, in the style of its precursor O’Haskell.

A particularly notable difference between Haskell and Timber is that Timber uses a *strict* evaluation order. This choice has primarily been motivated by a desire to facilitate more predictable execution times, but it also brings Timber closer to the efficiency of traditional execution models. Still, Timber retains the purely functional characteristic of Haskell, and also supports construction of recursive structures of arbitrary type in a declarative way.

The first public release of the Timber compiler was announced in December 2008. It uses the Gnu C compiler as its back-end and targets POSIX-based operating systems. Binary installers for Linux and MacOS X can be downloaded from the Timber web site

timber-lang.org. A bug-fix release (v 1.0.3) was made available in May 2009.

The current source code repository (also available on-line) includes a new way of organizing external interfaces, which separates access to OS, hardware or library services from the definition of a particular runtime system. This move greatly simplifies the construction of both external bindings and cross-compilation targets, which is utilized in on-going development of Xlib, OpenGL, iPhone as well as ARM7 support.

Other active projects include interfacing the compiler to memory and execution-time analysis tools, extending it with a supercompilation pass, and taking a fundamental grip on the generation of type error messages. The latter work will be based on principles developed for the Helium compiler (→ 2.2).

Further reading

<http://timber-lang.org>

3.2.5 Ur/Web

Report by:	Adam Chlipala
Status:	beta release

Ur/Web is a domain-specific language for building modern web applications. It is built on top of the **Ur** language as a custom standard library with special compiler support. Ur draws inspiration from a number of sources in the world of statically-typed functional programming. From Haskell, Ur takes purity, type classes, and monadic IO. From ML, Ur takes eagerness and a module system with functors and type abstraction. From the world of dependently-typed programming, Ur takes a rich notion of type-level computation.

The Ur/Web extensions support the core features of today’s web applications: “Web 1.0” programming with links and forms, “Web 2.0” programming with non-trivial client-side code, and interaction with SQL database backends. Considering programmer productivity, security, and scalability, Ur/Web has significant advantages over the mainstream web frameworks. Novel facilities for statically-typed metaprogramming enable new styles of abstraction and modularity. The type system guarantees that all kinds of code interpretable by browsers or database servers are treated as richly-typed syntax trees (along the lines of familiar examples of GADTs), rather than as “strings”, thwarting code injection attacks. The whole-program optimizing compiler generates fast native code which does not need garbage collection.

The open source toolset is in beta release now and should be usable for real projects. I expect the core feature set to change little in the near future, and the next few releases will probably focus on bug fixes and browser compatibility.

Further reading

<http://www.impredicative.com/ur/>

3.3 Type System / Program Analysis

3.3.1 Free Theorems for Haskell (and Curry)

Report by:	Janis Voigtländer
Participants:	Daniel Seidel, Jan Christiansen

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. Recent publications are “Taming Selective Strictness” (ATPS’09) and “Free Theorems for Functional Logic Programs” (PLPV’10). The latter, joint work with Jan Christiansen, considers the situation when moving from Haskell to Curry (\rightarrow 3.2.1).

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. 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://linux.tcs.inf.tu-dresden.de/~voigt/ft>. General 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

While the web-based tool is restricted to algebraic data types, type synonyms, and type classes from Haskell standard libraries, the shell-based tool also enables the user to declare their own algebraic data types and so on, and then to derive free theorems from types involving those. A distinctive feature of the web-based tool is to export the generated theorems in PDF format. By popular demand (≥ 1 person), now also the \LaTeX source for that PDF can be obtained as output.

Further reading

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

3.3.2 The Disciplined Disciple Compiler (DDC)

Report by:	Ben Lippmeier
Status:	alpha, active

See: <http://haskell.org/communities/05-2009/html/report.html#sect3.3.2>.

4 Tools

4.1 Transforming and Generating

4.1.1 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 which makes it easy to write *catamorphisms* (that is, functions that do to any datatype what *foldr* does to lists). You can define tree walks 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.

Idiomatic tree computations are neatly expressed in terms of copy, default, and collection rules. Computed results can masquerade as subtrees and be analyzed accordingly. 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.

The system is in use by a variety of large and small projects, such as the Utrecht Haskell Compiler UHC, the editor Proxima for structured documents, the Helium compiler (\rightarrow 2.2), the Generic Haskell compiler, and UUAG itself. The current version is 0.9.12 (October 2010), is extensively tested, and is available on Hackage.

We recently added support for building AG files through Cabal. A small Cabal plugin is installed upon installation of UUAG, which provides a userhook that deals with AG files and their dependencies.

Further reading

- <http://www.cs.uu.nl/wiki/bin/view/HUT/AttributeGrammarSystem>
- <http://hackage.haskell.org/package/uuagc-0.9.12>

4.1.2 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.

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".

Further reading

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

4.1.3 HFusion

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

HFusion is an experimental tool for optimizing Haskell programs. It is based on an algebraic approach where functions are internally represented in terms of a recursive program scheme known as *hylomorphism*. 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 struc-

tures produced in function compositions.

We offer a web interface to test the technique on user-supplied recursive definitions. The user can ask HFusion to transform a composition of two functions into an equivalent program which does not build the intermediate data structure involved in the composition. In future developments of the tool we plan to find fusable compositions within programs automatically.

In its current state, HFusion is able to fuse compositions of general recursive functions, including primitive recursive functions like `dropWhile` or `factorial`, 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).

```
Definitions
sumAcc :: Num a => a -> [a] -> a
sumAcc a [] = a
sumAcc a (x:xs) = sumAcc (a+x) xs

filter :: (a -> Bool) -> [a] -> [a]
filter p [] = []
filter p (a:ls) = if p a then a : filter p ls
                  else filter p ls

Functions to be fused
f: sumAcc (first function)
g: filter (second function)
 Check types

Result
f .n g = sf (result's name)
n = 2
 Hylo representation
Fuse!
```

```
Output
sf p a [] = a
sf p a (all:ls8) =
  if p all then sf p (a + all) ls8
  else sf p a ls8
```

Further reading

- Documentation about the tool can be found in [HFusion home](#)
- HFusion web interface is available from this [URL](#)

4.1.4 Optimus Prime

Report by:	Jason Reich
Participants:	Colin Runciman, Matthew Naylor
Status:	experimental

Optimus Prime is project developing a *supercompiler* for programs written in *F-lite*, the subset of Haskell used by the Reduceron (\rightarrow 2.6). It draws heavily on Neil Mitchell's work on the Supero supercompiler for YHC Core.

The project is still at the highly experimental stage but preliminary results are very encouraging. The process appears to produce largely *deforested* programs where higher-order functions have been *specialised*. This, as a consequence, appears to enable further

gains from mechanisms such as *speculative evaluation of primitive redexes* on the Reduceron architecture.

Optimus Prime supercompilation has led to a 74% reduction in the number of Reduceron clock-cycles required to execute some micro-examples.

Work continues on improving the execution time of the supercompilation transformation and improving the performance of the supercompiled programs.

Contact

<http://www.cs.york.ac.uk/people/?username=jason>

Further reading

<http://optimusprime.posterous.com/>

4.1.5 Derive

Report by:	Neil Mitchell
Status:	v2.0

The Derive tool is used to generate formulaic instances for data types. For example given a data type, the Derive tool can generate over 25 instances, including the standard ones (`Eq`, `Ord`, `Enum` etc.) and others such as `Binary` and `Functor`. Derive can be used with `SYB`, `Template Haskell` or as a standalone preprocessor. This tool serves a similar role to `DrIFT`, but with additional features.

Further reading

<http://community.haskell.org/~ndm/derive/>

4.1.6 lhs2TeX

Report by:	Andres Löh
Status:	stable, maintained

This tool by Ralf Hinze and Andres Löh is a pre-processor that transforms literate Haskell code into \LaTeX documents. The output is highly customizable by means of formatting directives that are interpreted by `lhs2TeX`. Other directives allow the selective inclusion of program fragments, so that multiple versions of a program and/or document can be produced from a common source. The input is parsed using a liberal parser that can interpret many languages with a Haskell-like syntax, and does not restrict the user to Haskell 98.

The program is stable and can take on large documents.

The current version is 1.14 and is soon going to be replaced with 1.15, a maintenance release that fixes some problems mainly with the Windows version, but will not introduce any new features.

Since version 1.14, `lhs2TeX` has an experimental mode for typesetting Agda code.

Further reading

<http://www.cs.uu.nl/~andres/lhs2tex>

4.2 Analysis and Profiling

4.2.1 SourceGraph

Report by:	Ivan Lazar Miljenovic
Status:	version 0.5.5.0

SourceGraph is a utility program aimed at helping Haskell programmers visualize their code and perform simple graph-based analysis (representing entities as nodes in the graphs and function calls as directed edges). It is a sample usage of the Graphalyze library (\rightarrow 5.7.2), which is designed as a general-purpose graph-theoretic analysis library. These two pieces of software were originally developed as the focus of my mathematical honors thesis, “Graph-Theoretic Analysis of the Relationships Within Discrete Data”.

Whilst fully usable, SourceGraph is currently limited in terms of input and output. It analyses all `.hs` and `.lhs` files recursively found in the provided directory, parsing most aspects of Haskell code (cannot parse Haskell code using CPP, HaRP, TH, FFI and XML-based Haskell code; difficulty parsing Data Family instances, unknown modules and record puns and wildcards). The results of the analysis are created in an HTML file in a “SourceGraph” subdirectory of the project’s root directory.

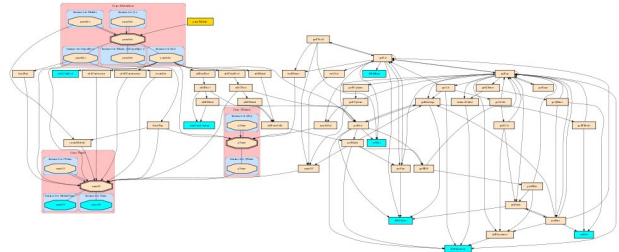
Changes since the previous release of HCAR include:

- Can now analyse a project from the Cabal file or an overall base module.
- Now parses data definitions and class/instance declarations.
- Usage of colour and shapes to indicate different types of entities and entity relationships with a legend to explain the differences.
- Removal of the Relative Neighbourhood clustering algorithm (as it didn’t provide much useful information).

Current analysis algorithms utilized include: alternative module groupings, whether a module should be split up, root analysis, clique and cycle detection as well as finding functions which can safely be compressed down to a single function. Please note however that SourceGraph is *not* a refactoring utility, and that its analyses should be taken with a grain of salt: for example, it might recommend that you split up a module,

because there are several distinct groupings of functions, when that module contains common utility functions that are placed together to form a library module (e.g., the Prelude).

Sample SourceGraph analysis reports can be found at http://code.haskell.org/~ivanm/Sample_SourceGraph/SampleReports.html. A tool paper on SourceGraph has been accepted for the ACM SIGPLAN 2010 Workshop on Partial Evaluation and Program Manipulation.



Further reading

- <http://hackage.haskell.org/package/SourceGraph>
- <http://ivanmiljenovic.files.wordpress.com/2008/11/honoursthesis.pdf>

4.2.2 HLint

Report by:	Neil Mitchell
Status:	v1.6

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. HLint can now spot where you have used a map or fold, gives many more hints, warns about redundant extensions etc. HLint is now used by the Darcs team and is one of the most popular applications on Hackage.

Further reading

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

4.2.3 hp2any

Report by:	Patai Gergely
Status:	experimental, on hold

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`.

Further reading

<http://www.haskell.org/haskellwiki/Hp2any>

4.3 Development

4.3.1 Hoogle — Haskell API Search

Report by:	Neil Mitchell
Status:	v4.0
See:	http://haskell.org/communities/05-2009/html/report.html#sect4.4.1 .

4.3.2 HEAT: The Haskell Educational Advancement Tool

Report by:	Olaf Chitil
Status:	active

Heat is an interactive development environment (IDE) for learning and teaching Haskell. Heat was designed for novice students learning the functional programming language Haskell. Heat provides a small number of supporting features and is easy to use. Heat is portable, small and works on top of the Haskell interpreter Hugs.

Heat provides the following features:

- Editor for a single module with syntax-highlighting and matching brackets.
- Shows the status of compilation: non-compiled; compiled with or without error.
- Interpreter console that highlights the prompt and error messages.

- If compilation yields an error, then the source line is highlighted and additional error explanations are provided.
- Shows a program summary in a tree structure, giving definitions of types and types of functions . . .
- Automatic checking of all (Boolean) properties of a program; results shown in summary.

Over the summer 2009 Heat was completely re-engineered to provide a simple and clean internal structure for future development. This new version still misses a few features compared to the current 3.1 version and hence a new release will only appear in 2010.

Further reading

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

4.3.3 HaRe — The Haskell Refactorer

Report by:	Simon Thompson
Participants:	Huiqing Li, Chris Brown, Chaddai Fouché, 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 fifth major release. HaRe supports full Haskell 98, and is integrated with Emacs (and XEmacs) 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, Huiqing's PhD thesis and Chris's

PhD thesis). The final report for the project appears there, too.

Chris Brown has recently passed his PhD; his PhD thesis entitled “Tool Support for Refactoring Haskell Programs” is available from our webpage.

Recent developments

- More structural and datatype-based refactorings have been studied by Chris Brown, including transformation between `let` and `where`, generative folding, introducing pattern matching, and introducing case expressions;
- Clone detection and elimination support has been added, to allow the automatic detection and semi-automatic elimination of duplicated code in Haskell.

Further reading

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

4.3.4 DarcsWatch

Report by:	Joachim Breitner
Status:	working

DarcsWatch is a tool to track the state of Darcs (→ 6.1.1) patches that have been submitted to some project, usually by using the `darcs send` command. It allows both submitters and project maintainers to get an overview of patches that have been submitted but not yet applied.

The DarcsWatch internals were changed during the Darcs hacking sprint in Vienna, to allow for a deeper integration into the roundup bug tracking instance used by the Darcs projects. Also, other extensions of darcs-watch are easier now. DarcsWatch continues to be used by the `xmonad` project (→ 6.1.2) and a few developers. At the time of writing, it was tracking 41 repositories and 2627 patches submitted by 162 users.

Further reading

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

4.3.5 HSFFIG

Report by:	Dmitry Golubovsky
Status:	release

Haskell FFI Binding Modules Generator (HSFFIG) is a tool which parses C include files (`.h`) and generates Haskell Foreign Functions Interface import declarations for all functions, `#define`'d constants (where possible), enumerations, and structures/unions (to access their members). It is assumed that the GNU C Compiler and

Preprocessor are used. Auto-generated Haskell modules may be imported into applications to access the foreign library's functions and variables.

HSFFIG has been in development since 2005, and was recently released on Hackage. The current version is 1.1.2 which is mainly a bug-fix release for the version 1.1.

The package provides a small library to link with programs using auto-generated imports, and two executable programs:

- `hsffig`: a filter program which reads pre-processed include files from standard input, and produces one large `.hsc` file;
- `ffipkg`: a program which automates the process of building a Cabal package out of C include files by the means of automated running `hsffig` and other tools necessary to build a Haskell package.

Further reading

- The HSFFIG package on Hackage
<http://hackage.haskell.org/package/HSFFIG>
- The HSFFIG Tutorial
<http://www.haskell.org/haskellwiki/HSFFIG/Tutorial>
- The FFI Imports Packaging Utility
http://www.haskell.org/haskellwiki/FFI_imports_packaging_utility

5 Libraries

5.1 Cabal and Hackage

Report by: Duncan Coutts

Background

Cabal is the Common Architecture for Building Applications and Libraries. It defines a common interface for defining and building Haskell packages. It is implemented as a Haskell library and associated tools which allow developers to easily build and distribute packages.

Hackage is a distribution point for Cabal packages. It is an online database of Cabal packages which can be queried via the website and client-side software such as `cabal-install`. Hackage enables end-users to download and install Cabal packages.

`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

There have been no new releases since the last HCAR, however there will soon be a release of Cabal-1.8 and a corresponding release of `cabal-install`. The primary change is that these releases will work with GHC 6.12. There is also a new “`cabal init`” command to help users create an initial “`.cabal`” file and a feature to maintain an HTML contents page for the Haddock documentation for all installed packages.

In addition there will be a minor release of `cabal-install-0.6.x` for users of GHC-6.10; the main change being a slight tweak in behaviour that should help with the general push to get packages to transition from base 3 to base 4.

Since the last report, `cabal-install` is now in wider use thanks to being included in the Haskell Platform (→ 5.2) and being packaged by various Linux distributions.

Hackage growth continues to be strong. For the second 6-month period in a row there has been 50% growth in the number of packages. There are now well over 1,500 different packages.

Looking forward

As ever, there are many improvements we want to make to Cabal, `cabal-install` and Hackage but our limiting factor is the amount of volunteer development time. We have over 100 open bugs and 150 open feature re-

quests. Many of these would require relatively little time for any competent Haskell hacker.

The other important project that people should consider helping out with is the new Hackage server implementation, based on Happstack. The current Hackage server has several widely acknowledged limitations. The new design should enable us to add many of the new features that users so frequently request. There has been some progress on this in the last few months but it needs some more work before we can consider a transition.

Further reading

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

5.2 Haskell Platform

Report by: Duncan Coutts

Background

The Haskell Platform (HP) is the name of a new “blessed” set of libraries and tools on which to build further Haskell libraries and applications. It takes the best packages from the more than 1500 on Hackage (→ 5.1). It is intended to provide a comprehensive, stable, and quality tested base for Haskell projects to work from.

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

Recent progress

We had the first major release of the platform earlier this year and followed that up with 2 further minor releases. The last of these uses GHC 6.10.4 and is available on Windows, OS X, as a generic Unix tarball and is included in several Linux distributions.

We established a steering committee to guide the discussions on the libraries mailing list to ensure that the necessary decisions do actually get made, recorded and communicated to the release engineering team.

The steering committee drafted a new procedure for adding packages to the Haskell Platform and this has now been ratified by the libraries mailing list. The new procedure involves writing a package proposal and discussing it on the libraries mailing list with the aim

of reaching a consensus. Details of the procedure are on the development wiki.

Looking forward

There had been a plan for a second major release of the platform using GHC 6.10, however that has now been shelved and the next major release will be in January and will use GHC 6.12. In the meantime, for people testing their packages with the GHC 6.12 release candidates, there will be a source-only beta version available.

Future major releases will be on a 6 month schedule. Major releases may include new and updated packages while minor releases will only contain bug fixes and fixes for packaging problems.

We would like to invite package authors to propose new packages for the upcoming major releases. We also invite the rest of the community to take part in the review process on the libraries mailing list libraries@haskell.org.

Further reading

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

5.3 Auxiliary Libraries

5.3.1 hmatrix

Report by:	Alberto Ruiz
Status:	stable, maintained

The *hmatrix* library is a purely functional interface to numerical linear algebra, internally implemented using GSL, BLAS, and LAPACK. The latest stable version is available from Hackage.

Recent work includes the experimental library *hTensor* (→ 5.3.2) for multidimensional arrays and simple tensor computations.

Further reading

<http://www.hmatrix.googlepages.com>

5.3.2 hTensor

Report by:	Alberto Ruiz
Status:	experimental, active development

hTensor is an experimental library for multidimensional arrays, oriented to support simple tensor computations and multilinear algebra. Array dimensions

(indices) are selected by name in expressions and Einstein's summation convention for repeated indices is automatically applied. We provide two main data types: *simple arrays* in which contractions only require equal dimension size, and *tensors*, whose indices are labeled as *covariant* or *contravariant* (subindex or superindex), and contractions can only be done on pairs of complementary indices. Arguments are automatically made conformant by replicating them along extra dimensions appearing in an operation. There is also preliminary support for geometric algebra and tensor decompositions.

The library has a purely functional interface: arrays are immutable, and operations typically work on whole structures which can be assembled and decomposed using simple primitives. It is built on top of *hmatrix* (→ 5.3.1), so coordinates are internally stored in C arrays and tensor products are implemented using BLAS. Therefore, big arrays can in principle be efficiently processed, although some functions are naively defined and not optimized. Consistency of dimension sizes is currently checked at run time. Future work includes static checking of conformability and a GUI for tensor diagrams.

A tutorial can be found in the project's web page.

Further reading

<http://perception.inf.um.es/tensor>

5.3.3 The Neon Library

Report by:	Jurriaan Hage
------------	---------------

As part of his master thesis work, Peter van Keeken implemented a library to data mine logged Helium (→ 2.2) programs to investigate aspects of how students program Haskell, how they learn to program, and how good Helium is in generating understandable feedback and hints. The software can be downloaded from <http://www.cs.uu.nl/wiki/bin/view/Hage/Neon>, which also gives some examples of output generated by the system. The downloads only contain a small sample of loggings, but it will allow programmers to play with it.

The recent news is that a paper about Neon will be published at SLE (1st Conference on Software Language Engineering), where it came under the heading of Tools for Language Usage.

On that note, there has been a posting by Simon Thompson, Sally Fincher and myself for a PhD student to work on understanding how students learn to program (in Haskell), in Kent. Also, recently I acquired a new master student to continue to the work of Peter van Keeken. One of this tasks will be to investigate the kind of parse errors students make, and continue to make. In the process, he shall add context properties (did the student pass or fail, what kind of programming background can we expect him or her to have) to our

database so that they can be employed by queries to increase external validity.

5.3.4 leapseconds-announced

Report by:	Björn Buckwalter
Status:	stable, maintained

The leapseconds-announced library provides an easy to use static LeapSecondTable with the leap seconds announced at library release time. It is intended as a quick-and-dirty leap second solution for one-off analyses concerned only with the past and present (i.e. up until the next as of yet unannounced leap second), or for applications which can afford to be recompiled against an updated library as often as every six months.

Version 2009 of leapseconds-announced contains all leap seconds up to 2009-01-01. A new version will be uploaded if/when the IERS announces a new leap second.

Further reading

- <http://hackage.haskell.org/cgi-bin/hackage-scripts/package/leapseconds-announced>
- <http://github.com/bjornbm/leapseconds-announced>

5.4 Parsing and Transforming

5.4.1 ChristmasTree

Report by:	Marcos Viera
Participants:	Doaitse Swierstra, Eelco Lempsink
Status:	experimental

See: <http://haskell.org/communities/05-2009/html/report.html#sect5.5.7>.

5.4.2 Utrecht Parser Combinator Library: New version

Report by:	Doaitse Swierstra
Status:	actively developed

The Utrecht Parser Combinator library has remained largely unmodified for the last five years, and has served us well. Over the years, however, new insights have grown, and with the advent of GADTs some internals could be simplified considerably. The Lernet summer school in February 2008 (<http://www.fing.edu.uy/inco/eventos/lernet2008/>) provided an incentive to start a rewrite of the library; a newly written tutorial will appear in the lecture notes, which will be published by Springer in the LNCS series. The text is also available as a technical report at <http://www.cs.uu.nl/research/techreps/UU-CS-2008-044.html>

Features

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

- Online result production, error recovery, combinators for parsing ambiguous grammars, an applicative interface, a monadic interface.
- Scanners can be switched dynamically, so several different languages can occur intertwined in a single input file.
- Fixes a potential black hole which went unnoticed for years in the code for the monadic bind as presented by Swierstra and Hughes in the ICFP 2003 paper: *Polish Parsers: Step by Step*.

A first version of the new library was recently released as the *wu-parsinglib* library, which has found its place in the *Text.ParserCombinators* category on Hackage.

Future plans

The final library, with an abstract interpretation part in order to get the parsing speed we got used to, will be release on Hackage again. We plan to extend the short tutorial which will appear in the LNCS series (45 pages) into a long tutorial.

Since many aspects of the old library, such as its applicative interface and the possibility to build e.g. parser for permutation phrases, have now come available elsewhere in other packages, we will also try to make the new library to conform as much as possible with these new developments.

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.

5.5 Mathematical Objects

5.5.1 dimensional: Statically checked physical dimensions

Report by:	Björn Buckwalter
Status:	active, mostly stable

See: <http://haskell.org/communities/11-2008/html/report.html#sect5.6.1>.

5.5.2 Halculon: units and physical constants database

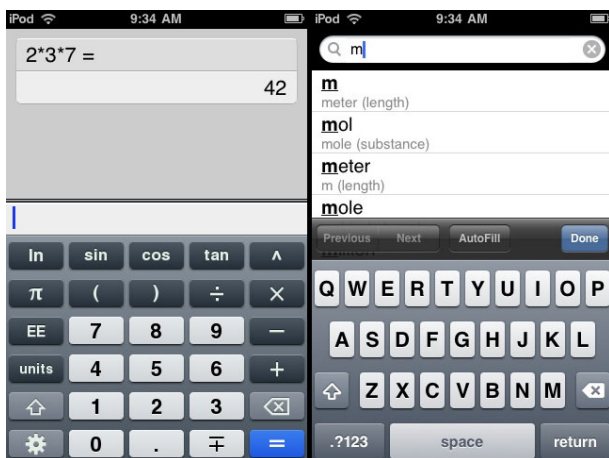
Report by:	Jared Updike
Status:	web application in beta, database stable

A number of Haskell libraries can represent numerical values with physical dimensions that are checked at runtime or compile time (including dimensional and the Numeric Prelude), but neither provide an exhaustive, searchable, annotated database of units, measures,

and physical constants. Halculon is an interactive unit database of 4,250 units, with a sample Haskell AJAX web application, based on the units database created by Alan Eliassen for the wonderful physical units programming language Frink. (Because each unit in Frink's unit.txt database is defined in terms of more basic unit definitions — an elegant approach in general — units.txt is inconvenient for looking up a single random unit; the entire file might need to be parsed to represent any given constant solely in terms of the base SI units, which is precisely what the Halculon database provides.)

Halculon also provides a carefully tuned, user- and developer-friendly search string database that aims to make interactive use pleasant. The database tables are available online and downloadable as UTF-8 text.

The example web application now has a mobile version available (tested in iPhone OS 3.1, Safari 3.0, and Firefox 2.0). For best results on the iPhone or iPod touch, Add to Home Screen to use the application in full screen. The calculator works offline, too.



Further reading

- o <http://www.updike.org/articles/Units>
- o <http://www.updike.org/halculon/>
- o <http://www.updike.org/halcmobile/>

5.5.3 Numeric prelude

Report by:	Henning Thielemann
Participants:	Dylan Thurston, Mikael Johansson
Status:	experimental, active development

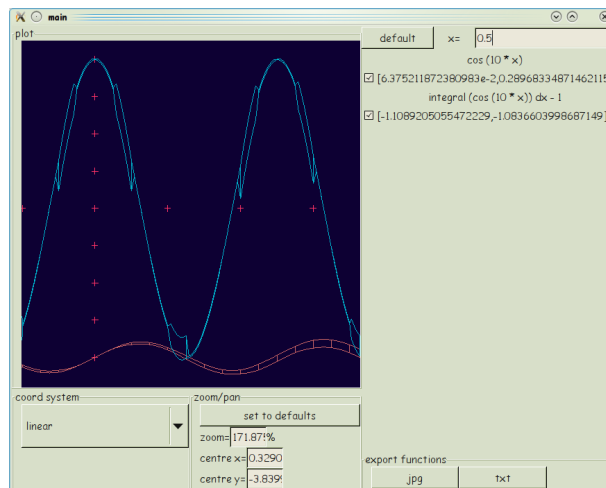
See: <http://haskell.org/communities/05-2009/html/report.html#sect5.6.2>.

5.5.4 AERN-Real and friends

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

AERN stands for Approximating Exact Real Numbers. We are developing a family of the following libraries for fast exact real number arithmetic:

- o AERN-Real: arbitrary precision safely rounded interval arithmetic with multiple backends (pure Haskell floating point numbers, MPFR, machine doubles) and with support for inner rounding, anti-consistent intervals and Kaucher arithmetic
- o AERN-RnToRm: arbitrary precision safely-rounded arithmetic of piece-wise polynomial function enclosures (PFEs) for functions over n-dimensional real intervals with support for inner rounding, anti-consistent intervals and approximated Kaucher arithmetic
- o AERN-RnToRm-Plot: GTK window for inspecting the graphs of PFEs in one variable (see figure below, showing a screenshot of an AERN-RnToRm-Plot window exploring an enclosure of $\cos(10x)$ (blue) and an enclosure of its primitive function (red))
- o AERN-Net: an implementation of distributed query-based (i.e., lazy) computation over analytical and geometrical objects



The development is driven mainly by the needs of our two research projects. We use the libraries extensively to:

- o prototype algorithms for reliable and ultimately converging methods for solving differential equations in many variables (AERN-RnToRm, AERN-Net)
- o solve numerical constraint satisfaction problems, especially those arising from verification of programs that use floating point numbers (AERN-RnToRm)

For our purposes AERN-Real has been stable for almost two years. It needs to be tested for a wider variety of applications before we can label it as stable. AERN-RnToRm is now also fairly stable thanks to a period of debugging and a comprehensive test suite. Nevertheless, it is rather slow as it has not been optimised

and there are occasional gaps in its functionality. The other libraries are even more experimental and incomplete. We recently added a useful mouse-driven zoom and pan feature to the function plot window.

The API of all the libraries is still occasionally changing but they provide a fairly extensive set of features and are reasonably well documented. The libraries are under active development and new features and bug fixes are expected to be submitted to Hackage in the coming 12 months. Notable planned additions in this period include:

- optimisations to the function enclosure arithmetic
- lazy communication of approximations of higher-order real functions using role switching
- infinite trees of enclosures for interval partial derivatives computed using automatic differentiation

Further reading

See Haddock documentation via Hackage — has links to research papers.

5.5.5 logfloat

Report by:	Wren Ng Thornton
Status:	stable?
Current release:	0.12.0.1
Portability:	GHC 6.8, GHC 6.10, Hugs Sept2006

The logfloat library provides a type for storing numbers in the log-domain. This is primarily useful for avoiding underflow when multiplying many small numbers in probabilistic models.

It also includes support for dealing with IEEE-754 floating point numbers (more) correctly, including: a class for types with representations for transfinite values, a class for partially ordered types, efficient and correct conversion from `Real` to `Fractional`, and bug fixes for Hugs' Prelude.

Future plans

Add a signed variant so negative numbers can also be projected into the log-domain.

Further reading

- Official source and documentation available on Hackage
- The development branch is available from <http://community.haskell.org/~wren/>

5.5.6 fad: Forward Automatic Differentiation

Report by:	Björn Buckwalter
Participants:	Barak A. Pearlmutter, Jeffrey Mark Siskind
Status:	active

Fad is an attempt to make as comprehensive and usable a forward automatic differentiation (AD) library as is possible in Haskell. Fad (a) attempts to be correct, by making it difficult to accidentally get a numerically incorrect derivative; (b) provides not only first-derivatives, but also a lazy tower of higher-order derivatives; (c) allows nested use of derivative operators while using the type system to reject incorrect nesting (perturbation confusion); (d) attempts to be complete, in the sense of allowing calculation of derivatives of functions defined using a large variety of Haskell constructs; and (e) tries to be efficient, in the sense of both the defining properties of forward automatic differentiation and in keeping the constant factor overhead as low as possible.

Version 1.0 of fad was uploaded to Hackage on April 3. Recent changes can be found via `git clone git://github.com/bjornbm/fad.git`

Further reading

- <http://github.com/bjornbm/fad>
- <http://flydynamikern.blogspot.com/2009/04/announce-fad-10-forward-automatic.html>

5.6 Data types and data structures

5.6.1 HList — a library for typed heterogeneous collections

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

HList is a comprehensive, general purpose Haskell library for typed heterogeneous collections including extensible polymorphic records and variants (\rightarrow 1.3). 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 now others (Alexandra Silva, Joost Visser: PURE.CoddFish project), have also used HList for type-safe database access in Haskell. HList-based Records form the basis of OOHaskell (<http://darcs.haskell.org/OOHaskell>). The HList library relies on common extensions of Haskell 98.

HList is being used in AspectAG, typed EDSL of attribute grammars, and in HaskellDB. There has been many miscellaneous changes related to the names of exposed modules, fixity declarations, making hMap and similar functions maximally lazy, improving error messages and documentation. A new, 0.2, release of HList incorporates the patches from HaskellDB developers Justin Bailey and Brian Bloniarz. The new release has a working regression test suite.

We are investigating the use of type functions provided in the new versions of GHC.

Further reading

- o HList: <http://homepages.cwi.nl/~ralf/HList/>
- o OOHaskell: <http://homepages.cwi.nl/~ralf/OOHaskell/>

5.6.2 bytestring-trie

Report by:	Wren Ng Thornton
Status:	active development
Current release:	0.1.4
Portability:	Haskell 98 + CPP

The bytestring-trie library provides an efficient implementation of “dictionaries” mapping strings to values, using big-endian patricia tries (like `Data.IntMap`). In general `Trie` is more efficient than `Map ByteString` because memory and work is shared between strings with common prefixes, though the specifics will vary depending on the distribution of keys.

Future plans

- o Min- and max-views for treating tries as priority queues.
- o Efficient intersection and difference functions.

Further reading

- o Official source and documentation available on Hackage.
- o The development branch is available from <http://community.haskell.org/~wren/>.

5.7 Data processing

5.7.1 MultiSetRewrite

Report by:	Martin Sulzmann
Status:	stable

See: <http://haskell.org/communities/05-2009/html/report.html#sect5.8.3>.

5.7.2 Graphalyze

Report by:	Ivan Lazar Miljenovic
Status:	version 0.8.0.0

The Graphalyze library is a general-purpose, fully extensible graph-theoretic analysis library, which includes functions to assist with graph creation and visualization, as well as many graph-related algorithms. Also included is a small abstract document representation, with a sample document generator utilizing Pandoc (\rightarrow 6.4.1). Users of this library are able to mix and match Graphalyze’s algorithms with their own. Changes since previous versions have focused on refining the contents of the library and inclusion of new analysis algorithms, with future plans to re-write the document generation modules to use pretty-printing functions.

Graphalyze is used in SourceGraph (\rightarrow 4.2.1) (which is the driving force behind improvements to Graphalyze), and was initially developed as part of my Mathematics Honours’ thesis, *Graph Theoretic Analysis of Relationships Within Discrete Data*. The focus on this thesis was to develop computational tools to allow people to analyze discrete data sets.

Further reading

- o <http://hackage.haskell.org/package/Graphalyze>
- o <http://ivanmiljenovic.files.wordpress.com/2008/11/honoursthesis.pdf>

5.8 Generic and Type-Level Programming

5.8.1 uniplate

Report by:	Neil Mitchell
------------	---------------

See: <http://haskell.org/communities/05-2009/html/report.html#sect5.9.1>.

5.8.2 Generic Programming at Utrecht University

Report by:	José Pedro Magalhães
Participants:	Stefan Holdermans, Johan Jeuring, Sean Leather, Andres Löh, Thomas van Noort
Status:	actively developed

One of the research themes investigated within the Software Technology Center in the Department of In-

formation and Computing Sciences at Utrecht University is generic programming. Over the last 10 years, we have played a central role in the development of generic programming techniques, languages, and libraries.

Currently, we are maintaining four generic programming libraries: `emgm`, `multirec`, `regular`, and `syb`. We report on the latter three in this entry; `emgm` has its own entry (\rightarrow 5.8.3).

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 current versions are 0.4 for `multirec` and 0.3 for `zipper`. There is ongoing development for `multirec`, and new releases are expected soon: we will likely add more functions and perhaps add a `multirec-extras` package as we just did for `regular` (see below). We are also working on support for datatypes with parameters and datatype compositions.

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 represent the pattern functor of the datatype to represent generically. We have recently released version 0.2 on Hackage, and also an extra package with more generic functions. Together they provide a number of typical generic programming examples, but also a number of less well-known but useful generic functions: `deep seq`, `QuickCheck`'s `arbitrary` and `coarbitrary`, and `binary`'s `get` and `put`.

syb Scrap Your Boilerplate (`syb`) has been supported by GHC since the 6.0 release. The 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. Recent work has allowed the separation of `syb` into a core module, `Data.Data`, which comes

with GHC on the `base` package, and the `syb` package, which is available on Hackage. This allows the development of the library independently of compiler releases. With GHC version 6.12 it will be possible to update the `syb` package, and we plan to release a new version with more functions and efficiency improvements.

We are also working on a new version of our library for generic rewriting. We have previously described the rewriting library in a paper and released it on Hackage as the `rewriting` package. Currently we are implementing support for families of (possibly mutually) recursive datatypes and preconditions.

Finally, we have been looking at benchmarking and improving the performance of different libraries for generic programming (\rightarrow 5.8.4).

Further reading

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

5.8.3 Extensible and Modular Generics for the Masses (EMGM)

Report by:	Sean Leather
Participants:	José Pedro Magalhães, Alexey Rodriguez, Andres Löh
Status:	actively developed

Extensible and Modular Generics for the Masses (EMGM) is a general-purpose library for generic programming with type classes.

Introduction

EMGM is a library for of datatype-generic programming using type classes. We represent Haskell datatypes as values using a sum-of-products structure representation. The foundation of EMGM allows programmers to write generic functions by induction on the structure of datatypes. The use of type classes in EMGM allows generic functions to support ad-hoc cases for arbitrary datatypes.

The library provides a sizable (and constantly growing) collection of ready-to-use generic functions. Here are some examples of these functions:

- `Crush`, a useful generalization of fold-like operations that supports flattening, integer operations, and logic operations on all values of an arbitrary datatype
- Extensible `Read` and `Show` functions to which one might add special cases for certain types
- `Collect` for collecting values of a certain type contained within a value of a different type
- `ZipWith`, a generic version of the standard `zipWith`

EMGM also comes with support for standard datatypes such as lists, `Either`, `Maybe`, and tuples. Adding support for your own datatype is straightforward using the `deriving API`.

Background

The ideas for EMGM come from research by Ralf Hinze, Bruno Oliveira, and Andres Löh. It was further explored in a comparison of generic programming libraries by Alexey Rodriguez, et al. Our particular implementation was developed simultaneously along with lecture notes for the 2008 Advanced Functional Programming Summer School. The article from these lectures has been extended and published as a [technical report](#).

Recent Development

No changes have been made since the previous report.

Future plans

We plan to continue developing EMGM and to explore the use of this library in many domains. There should be a major release before the next report. We welcome ideas or contributions from the community.

Contact

Let us know if you use EMGM, how you use it, and where it can be improved. Contact us on the [Generics mailing list](#).

Further reading

More information can be found on the [EMGM website](#). Download the package and browse the API at the [Hackage page](#).

5.8.4 Optimizing generic functions

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

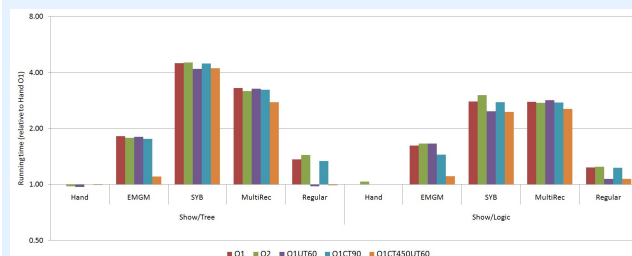
Datatype-generic programming increases program reliability by reducing code duplication and enhancing reusability and modularity. Several generic programming libraries for Haskell have been developed in the past few years. These libraries have been compared in detail with respect to expressiveness, extensibility, 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 more advanced optimization capabilities of GHC can be used to further optimize generic functions, sometimes achieving the same efficiency as hand-written code.

We have benchmarked four generic programming libraries: `emgm`, `syb`, `multirec`, and `regular`. We compare different generic functions in each of these libraries to a hand-written version. We have concluded that inlining plays a crucial role in the optimization of generics. In some cases, the inliner of GHC can already optimize a generic function up to the same performance of the hand-written version. However, this does not happen with simple `-O1` or `-O2` optimization levels: we need to tweak the unfolding flags to stimulate inlining.

As an example, we show the results of our benchmark for the generic `show` function below. We present how the libraries perform across different compiler optimizations: `UT60` stands for setting the `-funfolding-use-threshold` flag to 60 and `CT450` for setting `-funfolding-creation-threshold` to 450. We have benchmarked the function on two different datatypes: a simple binary tree (`Tree`) and a representation of logic expressions (`Logic`). The results are plotted relative to the hand-written version compiled with `-O1`, on a logarithmic scale of base 2. We can see that `syb` and `multirec` perform much worse than `emgm` and `regular`. More interestingly, though, is that by tweaking the inlining thresholds we can make `regular` as efficient as the hand-written version, and `emgm` almost as efficient.



Our benchmark provides many more interesting results, which can be seen in our [PEPM'10 paper](#). In the near future we plan to extend our benchmark and investigate the cases where simple tweaking of inlining flags is not sufficient to provide adequate performance.

Further reading

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

5.8.5 2LT: Two-Level Transformation

Report by:	Tiago Miguel Laureano Alves
Participants:	Joost Visser, Pablo Berdager, Alcino Cunha, José Nuno Oliveira, Hugo Pacheco
Status:	active

See: <http://haskell.org/communities/05-2009/html/report.html#sect5.9.6>.

5.8.6 Data.Label — “atoms” for type-level programming

Report by:	Claus Reinke
Status:	experimental

A common problem for type-level programming (extensible record libraries, type-level numbers, ...) in Haskell is where to define shared atomic types (record field labels, type tags, type numerals):

- identical types defined in separate modules are not compatible
- common imports defining common types for several projects hurt modularity
- SML-style parameterized modules and type-sharing are not directly available

Using Template Haskell, and QuasiQuotes in particular, we can now at least work around this issue, by splitting the atoms:-) `Data.Label` provides type letters and combinators for constructing typed “atoms” from these letters, as well as quasiquoting and `Show` instances to hide some of this internal structure.

```
*Main> [$1|label|]
label
*Main> :t [$1|label|]
[$1|label|] :: L1 :< (La :< (Lb :< (Le :< L1)))
```

This workaround lets users choose between shared or locally defined labels:

```
module A where
import Data.Label
data MyLabel
x = [$1|label|]
y = undefined::MyLabel

module B where
import Data.Label
data MyLabel
x = [$1|label|]
y = undefined::MyLabel

module C where
import Data.Label
import A
import B
ok = [A.x,B.x]
fails = [A.y,B.y]
```

It does so by offering a meta-level commonality: `A` and `B` do not have to agree on a common module to declare all their common types (`Data.Label` is unaffected by the specific labels its importers might use), they only need to agree on a common way of declaring all their sharable “atomic” types.

Further reading

- Example code: <http://community.haskell.org/~claus/misc/labels.hs>
<http://community.haskell.org/~claus/misc/Data/Label/TH.hs>
<http://community.haskell.org/~claus/misc/Data/Label.hs>
- Discussion: <http://www.haskell.org/pipermail/haskell-cafe/2009-April/059819.html>

5.9 User interfaces

5.9.1 Gtk2Hs

Report by:	Axel Simon
Participants:	Peter Gavin 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 and, of course, various desktops on Linux, Solaris, FreeBSD, and Mac OS X using X11.

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
- Cross-platform, multi-format multimedia playback with GStreamer
- Extensive reference documentation
- An implementation of the “Haskell School of Expression” graphics API
- Support for the Glade visual GUI builder
- Bindings to some Gnome extensions: gio, GConf, GtkSourceView 1.0 and 2.0
- An easy-to-use installer for Windows
- Packages for Fedora, Gentoo (→ 2.7.1), Debian, and FreeBSD
- New features added during the last months:
 - The terminal widget VTE
 - A binding the webkit to display web pages
 - More demos and more functions bound

The Gtk2HS library is continually being improved with new bindings, documentation, and bug fixes. Outside contributions are always welcome! We have recently re-surrected our code generator tool which means that new functions from Gtk+ 2.18.3 can be bound by copy-and-paste followed by a quick edit of the C documentation to reflect the Haskell version. This significantly lowers the burden to help us out, so please get in touch if you are interested.

Besides cabalizing Gtk2HS, we are working on concurrency support which turns out to be non-trivial in the face of GHC's concurrent garbage collector. We plan to release a new version of Gtk2HS before Christmas.

Further reading

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

5.9.2 HQK

Report by:	Wolfgang Jeltsch
Participants:	Thomas Mönicke
Status:	provisional

HQK is an effort to provide Haskell bindings to large parts of the Qt and KDE libraries. We have developed a generator which can produce binding code automatically. In addition, we have developed a small Haskell module for accessing object-oriented libraries in a convenient way. This module also supports parts of Qt's signal-slot mechanism. In contrast to the original C++-based solution, type correctness of signal-slot connections is checked at compile time with our library.

We plan to develop a HQK-based UI backend for the Functional Reactive Programming library Grapefruit (<http://haskell.org/communities/05-2009/html/report.html#sect6.5.1>).

Further reading

<http://haskell.org/haskellwiki/HQK>

5.10 Graphics

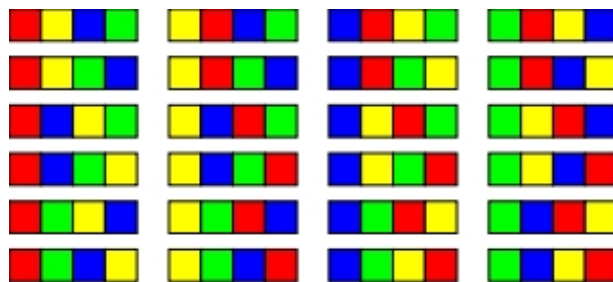
5.10.1 diagrams

Report by:	Brent Yorgey
Status:	active development

The diagrams library provides an embedded domain-specific language for creating simple pictures and diagrams. Values of type `Diagram` are built up in a compositional style from various primitives and combinators, and can be rendered to a physical medium, such as a file in PNG, PS, PDF, or SVG format. The overall vision

is for diagrams to become a viable alternative to DSLs like MetaPost or Asymptote, but with the advantages of being *purely functional* and *embedded*.

For example, consider the following diagram to illustrate the 24 permutations of four objects:



The diagrams library was used to create this diagram with very little effort (about ten lines of Haskell, including the code to actually generate permutations). The source code for this diagram, as well as other examples and further resources, can be found at <http://code.haskell.org/diagrams/>.

The library is currently undergoing a major rewrite, in order to use a more flexible constraint-solving layout engine and abstract out the rendering backend (the current version depends solely on the Cairo library for rendering). Other planned features include animation support, more sophisticated paths and path operations, and an `xmonad`-like core/contrib model for incorporating user-submitted extension modules.

Further reading

- <http://code.haskell.org/diagrams/>
- <http://byorgey.wordpress.com/2009/09/24/diagrams-0-2-1-and-future-plans/>
- <http://www.tug.org/metapost.html>
- <http://asymptote.sourceforge.net/>

5.10.2 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 (→ 6.11.7), and you can find a running example in the `lambdacube-bullet` 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>

5.10.3 GPipe

Report by: Tobias Bexelius
 Status: active development

GPipe models the entire graphics pipeline in a purely functional, immutable and type-safe way. It is built on top of the programmable pipeline (i.e., non-fixed function) of OpenGL 2.1 and uses features such as vertex buffer objects (VBO's), texture objects, and GLSL shader code synthetisation to create fast graphics programs. Buffers, textures, and shaders are cached internally to ensure fast framerate, and GPipe is also capable of managing multiple windows and contexts. GPipe's aim is to be as close to the conceptual graphics pipeline as possible, and not to add any more levels of abstraction.

In GPipe, you work with four main data types: PrimitiveStreams, FragmentStreams, FrameBuffers, and textures. They are all immutable, and all parameterized on the type of data they contain to ensure type safety between pipeline stages. By creating your own instances of GPipes type classes, it is possible to use additional data types on the GPU.

Version 1.0.1 with documentation is released on Hackage. Work continues on improving performance of GPipe, including adding support for mutable resources

in some way. I will also expand the wiki with more examples and tutorials. Any help would be appreciated!

Further reading

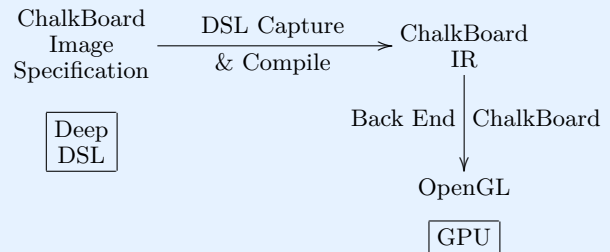
<http://www.haskell.org/haskellwiki/GPipe>

5.10.4 ChalkBoard

Report by: Andy Gill
 Participants: Kevin Matlage, Andy Gill
 Status: ongoing

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. The intention is that we will use ChalkBoard to animate educational videos, as well as processing streaming videos.

Here is the basic architecture of ChalkBoard.



The image specification language is a deeply embedded Domain Specific Language (DSL). We capture and compile our DSL, rather than interpret it directly. In order to do this, and allow use of a polygon-based back-end, we have needed to make some interesting compromises, but the language captured remains pure, has a variant of functors as a control structure, and has first-class images. We compile this language into an imperative intermediate representation that has first class *buffers* — regular arrays of colors or other entities. This language is then interpreted by macro-expanding each intermediate representation command into a set of OpenGL commands. In this way, we leverage modern graphics boards to do the heavy lifting of the language.

A release is planned for early November, and will be available on Hackage.

The new video processing technology will be premiered at PEPM'10.

Further reading

<http://www.ittc.ku.edu/csdl/ChalkBoard>

5.10.5 graphviz

Report by:	Ivan Lazar Miljenovic
Status:	version 2999.6.0.0

The graphviz library provides Haskell bindings to the *GraphViz* suite of tools for visualising graphs. This library was originally written by Matthew Sackman, but has been maintained (and almost completely re-written) by myself since the 2009.5.1 release.

The library supports almost all GraphViz attributes, and can produce and parse graphs represented in GraphViz's *Dot* language (albeit with the limitation of strict ordering of *Dot* statements). Clusters are also supported, and strings used for labels and identifiers are automatically quoted appropriately.

Currently supported are conversions from FGL graphs as well as annotating nodes and edges in FGL graphs with their positions, etc. Support for other graph types is planned for a future release.

For a sample graph visualised using the graphviz library, see SourceGraph (→ 4.2.1).

Further reading

- <http://hackage.haskell.org/package/graphviz>
- <http://www.graphviz.org/>

5.11 Music

5.11.1 Haskore revision

Report by:	Henning Thielemann
Participants:	Paul Hudak
Status:	experimental, active development

See: <http://haskell.org/communities/05-2009/html/report.html#sect5.12.1>.

5.11.2 Euterpea

Report by:	Paul Hudak
Participants:	Eric Cheng, Paul Liu, Donya Quick
Status:	experimental, active development

Euterpea is a new Haskell library for computer music applications. It is a descendent of Haskore and HasSound, and is intended for both educational purposes as well as serious computer music development. Euterpea is a “wide-spectrum” library, suitable for high-level music representation, algorithmic composition, and analysis; mid-level concepts such as MIDI; and low-level audio processing, sound synthesis, and instrument design. It also includes a “musical user interface”, a set of computer-music specific GUI widgets such as keyboards, guitar frets, knobs, sliders, and so on. The performance of Euterpea is intended to be as good or better than any existing computer music language – it can be used for real-time applications, not just using

MIDI, but also using a high-performance back-end for real-time audio.

Euterpea is being developed at Yale in Paul Hudak's research group, where it has become a key component of Yale's new Computing and the Arts major. Hudak is teaching a two-term sequence in computer music using Euterpea, and is developing considerable pedagogical material, including a new textbook tentatively titled “The Haskell School of Music”. The name “Euterpea” is derived from “Euterpe”, who was one of the nine Greek Muses (goddesses of the arts), specifically the Muse of Music.

History

Haskore is a Haskell library developed almost 15 years ago by Paul Hudak at Yale for high-level computer music applications. HasSound is a more recent Haskell library developed at Yale that serves as a functional front-end to csound's sound synthesis capabilities. Haskore and HasSound have evolved in a number of different ways over the years, most notably through Henning Thielemann's darcs library for Haskore, to which many people have contributed. There are many good ideas in that library, but it has become overly complex and lacks a coherent design concept.

Future plans

The Euterpea developers' plan is to shamelessly steal good ideas from these previous efforts, integrate them into a coherent new framework, remove dependencies from as many non-Haskell libraries as possible, add new features such as musical GUI widgets, and incorporate new methods for high-performance stream processing recently developed at Yale, to make Euterpea the library of choice for discriminating computer music hackers.

Further reading

- <http://plucky.cs.yale.edu/cs431>
- <http://plucky.cs.yale.edu/cs431/HaskoreSoeV-0.12.pdf>

5.12 Web and XML programming

5.12.1 Haskell XML Toolbox

Report by:	Uwe Schmidt
Status:	seventh major release (current release: 8.3.2)

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 (<http://haskell.org/communities/05-2009/html/report.html#sect5.13.2>) 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 consists of two packages, the old first approach (`hxt-filter`) based on filters and the newer and more flexible and safe approach using arrows (`hxt`). The old package `hxt-filter`, will further be maintained to work with the latest `ghc` version, but new development will only be done with the arrow based `hxt` package.

Features

- Validating XML parser
- Very liberal HTML parser
- Lightweight lazy parser for XML/HTML based on TagSoup (→ 5.12.2)
- 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`
- Tested with W3C XML validation suite
- Example programs
- Relax NG schema validator
- An HXT Cookbook for using the toolbox and the arrow interface
- Basic XSLT support
- Darcs repository with current development version (8.3.1) under <http://darcs2.fh-wedel.de/hxt>

Current Work

Currently mainly maintenance work is done. This includes space and runtime optimizations.

The HXT library is extensively used in the Holumbus project (→ 6.3.1), there it forms the basis for the index generation.

Further reading

The Haskell XML Toolbox Web page (<http://www.fh-wedel.de/~si/HXmlToolbox/index.html>) includes downloads, online API documentation, a cookbook with nontrivial examples of XML processing using arrows and RDF documents, and master theses

describing the design of the toolbox, the DTD validator, the arrow based Relax NG validator, and the XSLT system.

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 datatypes is described in another Wiki page (http://www.haskell.org/haskellwiki/HXT/Conversion_of_Haskell_data_from/to_XML).

5.12.2 tagsoup

Report by:

Neil Mitchell

TagSoup is a library for extracting information out of unstructured HTML code, sometimes known as `tagsoup`. The HTML does not have to be well formed, or render properly within any particular framework. This library is for situations where the author of the HTML is not cooperating with the person trying to extract the information, but is also not trying to hide the information.

The library provides a basic data type for a list of unstructured tags, a parser to convert HTML into this tag type, and useful functions and combinators for finding and extracting information. The library has seen real use in an application to give Hackage (→ 5.1) listings, and is used in the next version of Hoogle (→ 4.3.1).

Work continues on the API of `tagsoup`, and the implementation. In particular the development version is based around the HTML 5 specification, and supports many flavours of `ByteString` in addition to `String`.

Further reading

<http://community.haskell.org/~ndm/tagsoup>

6 Applications and Projects

6.1 For the Masses

6.1.1 Darcs

Report by:	Eric Kow
Participants:	Reinier Lamers, Ganesh Sittampalam
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 major release, Darcs 2.3, was in July 2009. This release provides better performance and easier installation. The next release promises to be a particularly exciting one:

1. **Faster repository-local operations:** Darcs developer Petr Rockai has successfully completed his 2009 Google Summer of Code program. This work optimizes Darcs' use of hashed storage repositories, making Darcs faster and more scalable in repository-local operations. We are extremely grateful for generous support of the Haskell.org mentoring organization, which has provided us with one of their Google Summer of Code slots.
2. **Hunk-splitting:** Ganesh Sittampalam has implemented an improvement that gives users much greater control over patches created by Darcs. In the interactive darcs record interface, users can opt to split hunk patches into smaller pieces, allowing for more granular patches where appropriate and making cherry picking more useful in practice.

These changes and more will appear in the upcoming Darcs 2.4 release, scheduled for January 2010. We still have a lot progress to make and are always open to contributions. Haskell hackers, we need your help!

Darcs is free software licensed under the GNU GPL. 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>

6.1.2 xmonad

Report by:	Don Stewart
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.

The new release 0.7 of xmonad added full support for the GNOME and KDE desktops, and adoption continues to grow, with binary packages of xmonad available for all major distributions.

Further reading

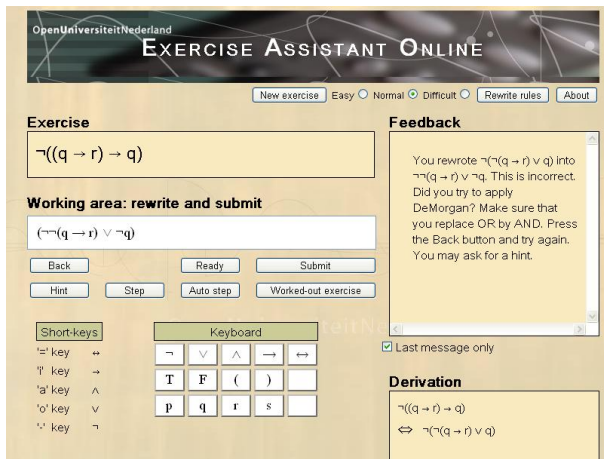
- Homepage: <http://xmonad.org/>
- Darcs source: `darcs get http://code.haskell.org/xmonad`
- IRC channel: `#xmonad @ irc.freenode.org`
- Mailing list: `<xmonad@haskell.org>`

6.2 Education

6.2.1 Exercise Assistants

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

At the Open Universiteit Nederland and Universiteit Utrecht we are continuing our work on tools that support students in solving exercises incrementally by checking intermediate steps. The distinguishing feature of our tools is the detailed feedback that they provide, on several levels. For example, we have an online exercise assistant that helps to rewrite logical expressions into disjunctive normal form. Students get instant feedback when solving an exercise, and can ask for a hint at any point in the derivation. Other areas covered by our tools are solving equations, reducing matrices to echelon normal form, and simplifying expressions in relation algebra.



We have just started to explore exercise assistants for learning how to program in Haskell. A case study was performed to use programming strategies for automatically assessing student programs submitted for a first-year course on functional programming in Utrecht.

For each exercise domain, we need the same functionality, such as unifying and rewriting terms, generating exercises, traversing terms, and testing for (top-level) equality of two terms. For these parts we are currently using the generic programming libraries `Uniplate` and `Multirec`, which help us to reduce code size and improve the reliability of our code. We have reported our experiences with [generic programming for domain reasoners](#), and identified some missing features in the libraries. Fully exploiting generic programming techniques is ongoing work.

We have recently integrated our tools with the Digital Mathematics Environment (DWO) of the Freudenthal Institute. This environment contains a collection of applets for practicing exercises in mathematics. A selected number of applets has been extended with our facility to automatically generate hints and worked-out examples, and the first results are promising. To offer this service, we have introduced [views for mathematical expressions](#) (based on the views proposed by Wadler), and combined these with our rewriting technology. A view specifies a canonical form, and abstracts over a set of algebraic laws. Our [feedback services](#) have recently been released as a Cabal source package.

Further reading

- Online exercise assistant, accessible from our [project page](#).
- Bastiaan Heeren and Johan Jeuring. [Canonical Forms in Interactive Exercise Assistants](#). Proceedings of Mathematical Knowledge Management (MKM 2009).
- Johan Jeuring, José Pedro Magalhães, and Bastiaan Heeren. [Generic Programming for Domain Reasoners](#). To appear in the Symposium on Trends in Functional Programming (TFP 2009).

- Alex Gerdes, Johan Jeuring, and Bastiaan Heeren. [Using Strategies for Assessment of Programming Exercises](#). To appear in the Technical Symposium on Computer Science Education (SIGCSE 2010).

6.2.2 Holmes, plagiarism detection for Haskell

Report by:	Jurriaan Hage
Participants:	Brian Vermeer

Years ago, Jurriaan Hage developed Marble to detect plagiarism among Java programs. Marble was written in Perl, takes just 660 lines of code and comments, and does the job well. The techniques used there, however, do not work well for Haskell, which is why a master thesis project was started, starring Brian Vermeer as the master student, to see if we can come up with a working system to discover plagiarism among Haskell programs. We are fortunate to have a large group of students each year that try their hand at our functional programming course (120-130 per year), and we have all the loggings of Helium that we hope can help us tell whether the system finds enough plagiarism cases. The basic idea is to implement as many metrics as possible, and to see, empirically, which combination of metrics scores well enough for our purposes. The implementation will be made in Haskell. One of the things that we are particularly keen about, is to make sure that for assignments in which students are given a large part of the solution and they only need to fill in the missing parts, we still obtain good results.

We are currently at the stage that metrics can be implemented on top of the Helium front-end. Many of these metrics will be defined on an auxiliary structure, the function-call flow graph. Dead-code removal has taken place, fully qualified names are used throughout, and template removal is now easily possible.

6.2.3 INblobs — Interaction Nets interpreter

Report by:	Miguel Vilaca
Participants:	Daniel Mendes
Status:	active
Portability:	portable (depends on wxHaskell)

See: <http://haskell.org/communities/05-2009/html/report.html#sect6.2.4>.

6.2.4 Yahc

Report by:	Miguel Pagano
Participants:	Renato Cherini
Status:	experimental, maintained

The first course on algorithms in CS at *Universidad Nacional de Córdoba* is centered on the derivations of algorithms from specifications, as proposed by R.S. Bird (*Introduction to functional programming using Haskell*, Prentice Hall Series in Computer Science, 1998), E.W. Dijkstra (*A Discipline of Programming*, Prentice Hall, 1976), and R.R. Hoogerwoord (*The design of functional*

programs: a calculational approach, Technische Universiteit Eindhoven, 1989). To achieve this goal, students should acquire the ability to manipulate complex predicate formulae; thus the students first learn how to prove theorems in a propositional calculus similar to the equational propositional logic of D. Gries and F.B. Schneier (*A Logical Approach to Discrete Math*, Springer-Verlag, 1993).

During the semester students make many derivations as exercises and it is helpful for them to have a tool for checking the correctness of their solutions. Yahc checks the correctness of a sequence of applications of some axioms and theorems to the formulae students are trying to prove. The student starts a derivation by entering an initial formula and a goal and then proceeds by telling Yahc which axiom will be used and the expected outcome of applying the axiom as a rewrite rule; if that rewriting step is correct then the process continues until the student reaches the goal.

At this moment the tool is in an early stage and we only consider propositional connectives (some of them associative-commutative). We expect to extend Yahc for allowing the resolution of logical puzzles. In the long term we are planning to consider an equational calculus with functions defined by induction over lists and natural numbers.

Further reading

<http://www.cs.famaf.unc.edu.ar/~mpagano/yahc/>

6.2.5 grolprep

Report by:	Dino Morelli
Participants:	Betty Diegel
Status:	experimental, actively developed

grolprep is a web application for studying the FCC GROL questions in preparation of taking the exams.

The study of this multiple-choice data is in the flash-card style. Students can choose from Elements 1, 3 and 8 and can specify any subelement of those three for specific study. Questions and answers can be randomly presented.

Additionally, simulations of the randomly-chosen exams can be practiced with this software.

grolprep will shortly be used by students of Avionics program at the Burlington Aviation Technology School.

Further reading

- o Live website: <http://ui3.info/grolprep/bin/fcc-grol-prep.cgi>
- o Project page: <http://ui3.info/d/proj/grolprep.html>
- o Source repository: `darcs get http://ui3.info/darcs/grolprep`

6.3 Web Development

6.3.1 Holumbus Search Engine Framework

Report by:	Uwe Schmidt
Participants:	Timo B. Hübel, Sebastian Reese, Sebastian Schlatt, Stefan Schmidt, Björn Peemöller, Stefan Roggensack, Alexander Treptow
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 web interface for Hayoo! is implemented with the Janus web server, written in Haskell and based on HXT (\rightarrow 5.12.1).

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.

- o The Holumbus Search Engine
- o The Holumbus Distribution Library
- o The Holumbus Storage System
- o 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

- o Highly configurable crawler module for flexible indexing of structured data
- o Customizable index structure for an effective search
- o *find as you type* search
- o Suggestions

- Fuzzy queries
- Customizable result ranking
- Index structure designed for distributed search
- Darcs repository with current development version under <http://darcs2.fh-wedel.de/holumbus>
- Distributed building of search indexes

Current Work

The indexer and search module will be used and extended to support the Hayoo! engine for searching the hackage package library (<http://holumbus.fh-wedel.de/hayoo/hayoo.html>).

In a follow-up project of Stefan Schmidt's system, Sebastian Reese is working on a *cookbook* for programming with the MapReduce framework and for giving tuning and configuration hints. Benchmarks for various small problems and for generating search indexes have shown that the architecture scales very well. The thesis with the final results will be finished end of March 2010.

In a further subproject, the so called Hawk system, Björn Peemöller and Stefan Roggensack developed a web framework for Haskell. The results of this master thesis will be published soon on the Holumbus web site.

The Hawk system is comparable in functionality and architecture with Ruby on Rail and other web frameworks. Its architecture follows the MVC pattern. It consists of a simple relational database mapper for persistent storage of data and a template system for the view component. This template system has two interesting features: First the templates are valid XHTML documents. The parts where data has to be filled in are marked with Hawk specific elements and attributes. These parts are in a different namespace, so they do not destroy the XHTML structure. The second interesting feature is that the templates contain type descriptions for the values to be filled in. This type information enables a static check whether the models and views fit together.

The Hawk framework is independent of the Holumbus search engine. It will be applicable for the development of arbitrary web applications, but one of the first applications is the reimplemention of the web interface for the Hayoo! search engine. This project will be done by Alexander Treptow within the next six months.

Further reading

The Holumbus web page (<http://holumbus.fh-wedel.de/>) includes downloads, Darcs 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 Schlatt'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>.

6.3.2 HCluster

Report by:	Alberto Gómez Corona
------------	----------------------

HCluster (provisional name) is a remote clustering middleware aimed initially at verifying online and offline computations in distributed electoral processes. Extended to permit clustering with distributed transactions and cloud computing.

- distributed transactions between connected nodes in the Internet
- work with online nodes as well as offline + synchronization
- hot plug-in of nodes
- no single point of failure/control
- theoretical massive scalability, reliability, availability

Any node can initiate a process (that may involve a transaction, a query, a calculation etc.). The design of synchronization permits nodes to work in online as well as offline mode with periodic synchronization with certain restrictions. The restrictions depend on algebraic properties of the transactions.

Distribution of data and distributed transactions are possible. The distribution is transparent to the programmer, so re-locations of data can be done among the nodes.

Finished basic services: HTTP protocol, reconnection, synchronization. Testing synchronization and online clustering now defined and coded the model for distributed transactions.

Future plans

- test distributed transactions
- create internet documentation

Contact

agocorona@gmail.com

6.3.3 JavaScript Monadic Writer

Report by:	Dmitry Golubovsky
Status:	active development

JavaScript Monadic Writer (JSMW) is an extensible monadic framework on top of the Haskell DOM bindings. It provides an EDSL to encode JavaScript statements and expressions in typesafe manner. It borrows

some ideas from HJScript, but uses slightly different EDSL notation.

The idea behind JSWM is to provide an intermediate form that could be used as an end-point for conversion from Haskell Core. The EDSL however may be considered as a programming tool on its own. While the EDSL alone is not sufficient for translation of an arbitrary Haskell program to JavaScript, Haskell type system is still available to help produce correct code.

Further reading

The `jsmw` package on Hackage: <http://hackage.haskell.org/cgi-bin/hackage-scripts/package/jsmw>

6.3.4 Haskell DOM Bindings

Report by:	Dmitry Golubovsky
Status:	active development

Haskell DOM bindings is a set of monadic smart constructors on top of the WebBits representation of JavaScript syntax to generate JavaScript code that calls DOM methods and accesses DOM objects' attributes.

In order to represent the hierarchy of DOM interfaces, Haskell type classes are used. For example, for the interfaces `Node` and `Document` (the latter inherits from the former) there are two classes: `CNode` and `CDocument`. Also, for each DOM interface, a phantom data type is defined: `TNode`, and `TDocument` in this case. Phantom types represent concrete values (references to DOM objects) while type classes are used for type constraints in functions working with DOM objects. The `CDocument` class is defined as:

```
class CNode a => CDocument a
data TNode
data TDocument
instance CNode TNode
instance CDocument TDocument
instance CNode TDocument
```

Type constraints are used to define methods of each class, e. g.

```
hasChildNodes :: (Monad mn, CNode this)
=> Expression this -> mn (Expression Bool)
```

so, `hasChildNodes` can be called on both `Node` and `Document`, but

```
createElement :: (Monad mn, CDocument this,
                  CElement zz)
=> Expression String -> Expression this
-> mn (Expression zz)
```

only on nodes representing documents.

The bindings were auto-generated from OMG IDL files provided by the Web Consortium. The IDL to

Haskell converter is based on H/Direct IDL parser. Automatic IDL conversion is expected to simplify Haskell Web development because of the large number of methods and attributes defined in contemporary DOM whose type signatures are hard and time-consuming to derive manually.

Further reading

- o Document Object Model (DOM) Technical Reports <http://www.w3.org/DOM/DOMTR>
- o The DOM package on Hackage <http://hackage.haskell.org/cgi-bin/hackage-scripts/package/DOM>

6.3.5 gitit

Report by:	John MacFarlane
Participants:	Gwern Branwen, Simon Michael, Henry Laxen, Anton van Straaten, Robin Green, Thomas Hartman, Justin Bogner, Kohei Ozaki, Dmitry Golubovsky, Anton Tayanovskyy, Dan Cook
Status:	active development

Gitit is a wiki built on Happstack and backed by a git or darcs filestore. Pages and uploaded files can be modified either directly via the VCS's command-line tools or through the wiki's web interface. Pandoc (→ 6.4.1) is used for markup processing, so pages may be written in (extended) markdown, reStructuredText, LaTeX, HTML, or literate Haskell, and exported in ten different formats, including LaTeX, ConTeXt, DocBook, RTF, OpenOffice ODT, and MediaWiki markup.

Notable features of gitit include:

- o Plugins: users can write their own dynamically loaded page transformations, which operate directly on the abstract syntax tree.
- o Math support: LaTeX inline and display math is automatically converted to MathML, using the `texmath` library.
- o Highlighting: Any git or darcs repository can be made a gitit wiki. Directories can be browsed, and source code files are automatically syntax-highlighted. Code snippets in wiki pages can also be highlighted.
- o Library: Gitit now exports a library, `Network.Gitit`, that makes it easy to include a gitit wiki (or wikis) in any Happstack application.
- o Literate Haskell: Pages can be written directly in literate Haskell.

Further reading

<http://gitit.net> (itself a running demo of gitit)

6.4 Data Management and Visualization

6.4.1 Pandoc

Report by: John MacFarlane
Participants: Andrea Rossato, Peter Wang, Paulo Tanimoto
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, and reStructuredText, and it can write markdown, reStructuredText, HTML, DocBook XML, OpenDocument XML, ODT, RTF, groff man, MediaWiki markup, GNU Texinfo, LaTeX, ConTeXt, and S5. Pandoc's markdown syntax includes extensions for LaTeX math, tables, definition lists, footnotes, and more.

Since the last report, there has been one release (1.2.1).

- Users may notice a significant speedup in reading markdown in `--smart` mode; the abbreviations parser has been made much more efficient.
- Default HTML output now wraps sections in divs with unique identifiers. This should aid manipulation using javascript and other tools.
- We have made some progress in replacing the old POSIX shell script wrappers with more portable Haskell wrappers.

Further reading

<http://johnmacfarlane.net/pandoc/>

6.4.2 HaExcel — From Spreadsheets to Relational Databases and Back

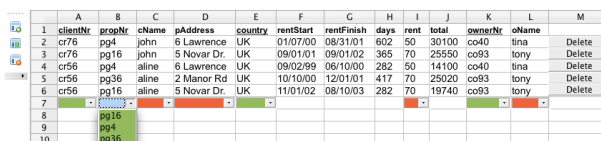
Report by: Jácome Cunha
Participants: João Saraiva, Joost Visser
Status: unstable, work in progress

HaExcel is a framework to manipulate and transform spreadsheets. It is composed by a generic/reusable library to map spreadsheets into relational database models and back: this library contains an algebraic data type to model a (generic) spreadsheet and functions to transform it into a relational model and vice versa. Such functions implement the refinement rules introduced in paper “From Spreadsheets to Relational Databases and Back”. The library includes two code generator functions: one that produces the SQL code to create and populate the database, and a function that generates Excel/Gnumeric code to map the database back into a spreadsheet. A MySQL database can also be created and manipulated using this library under HaskellDB.

The tool also contains a front-end to read spreadsheets in the Excel and Gnumeric formats: the front-end reads spreadsheets in portable XML documents using the *UMinho Haskell Libraries*. We reuse the spatial logic algorithms from the UCheck project to discover the tables stored in the spreadsheet.

Finally, two spreadsheet tools are available: a batch and an online tool that allows the users to read, transform and refactor spreadsheets.

Using part of HaExcel, we developed an OpenOffice Calc (<http://www.openoffice.org/product/calc.html>) add-on. Its back-end reuses part of HaExcel and its front-end is written in OpenOffice Basic. This add-on allows the integration of a relational model into the spreadsheet. Using this model the user gets three new features in the spreadsheet environment: *auto-completion* of columns, that is, choosing values of some columns, other columns become automatically completed; *safe deletion* of rows where the user is warned when deleting important information; and *no edition* of columns that could compromise the data integrity. All the features can be enabled and disabled by the user at any time. A snapshot of a spreadsheet with the add-on can be seen below.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	clientNr	propNr	cName	pAddress	country	rentStart	rentFinish	days	rent	total	ownerNr	oName	
2	cr76	pg4	john	6 Lawrence	UK	01/07/00	08/31/01	602	50	30100	co40	tina	Delete
3	cr76	pg16	john	5 Novar Dr.	UK	09/01/01	09/01/02	365	70	25550	co93	tony	Delete
4	cr56	pg4	aline	6 Lawrence	UK	09/02/99	06/10/00	282	50	14100	co40	tina	Delete
5	cr56	pg36	aline	2 Manor Rd	UK	10/10/00	12/01/01	417	70	25020	co93	tony	Delete
6	cr56	pg16	aline	5 Novar Dr.	UK	11/01/02	08/10/03	282	70	19740	co93	tony	Delete
7													
8		pg16											
9		pg4											
10		pg36											

More about this can be read in the paper “Discovery-based Edit Assistance for Spreadsheets”.

The sources, the online tool and the add-on are available from the project home page.

We are currently exploring foreign key constraints from their detection to their migration to the generated spreadsheet.

Further reading

<http://www.di.uminho.pt/~jacome>

6.4.3 SdfMetz

Report by: Tiago Miguel Laureano Alves
Participants: Joost Visser
Status: stable, maintained

See: <http://haskell.org/communities/05-2009/html/report.html#sect6.4.5>.

6.4.4 The Proxima 2.0 generic editor

Report by: Martijn Schrage
Participants: Lambert Meertens, Doaitse Swierstra
Status: actively developed

Proxima 2.0 is an open-source web-based version of the Proxima generic presentation-oriented editor for structured documents.

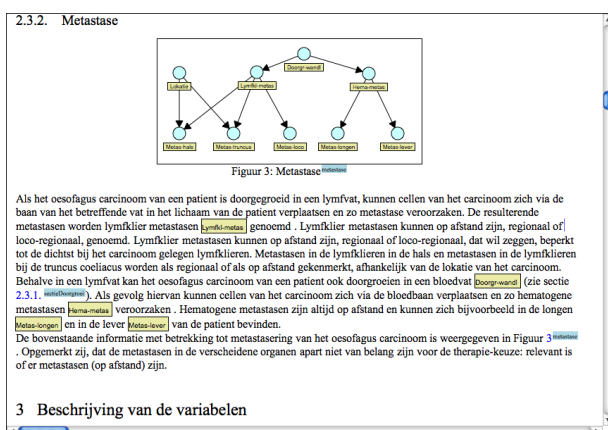
- Proxima is a *generic* editor. This means that the editor can be instantiated for arbitrary document types, supplemented by parser and presentation sheets. The content of a Proxima document can be mixed text, images and diagrams.
- Proxima is a *presentation-oriented editor*. This means that the user performs the edit operations on the WYSIWYG screen presentation of the document.
- Proxima is aware of the structure of the document. Even while editing the presentation of the document, the edit operations can be structural. For example, a section can be changed into a subsection.

Another feature of Proxima is that it offers generic support for specifying content-dependent computations. For example, it is possible to create a table of contents of a document that is automatically updated as chapters or sections are added or modified.

Proxima 2.0

Proxima 2.0 provides a web-interface for Proxima. Instead of rendering the edited document onto an application window, Proxima 2.0 is a web-server that sends an HTML rendering of the document to a client. The client catches mouse and keyboard events, and sends these back to the server, after which the server sends incremental rendering updates back to the client. As a result, advanced editors can be created, which run in any browser.

In the past half year, the system has been extended with drag and drop, improved navigation, session handling, and a number of techniques for handling network latency. Furthermore, a number of example editors have been implemented. The screenshot shows an editor for documenting Bayesian networks. It is running in a Firefox browser.



Future plans

Proxima 2.0 is an open source project. We are looking for people who would like to participate in the project.

Further reading

<http://www.oblomov.biz/proxima2.0.html>

6.5 Functional Reactive Programming

6.5.1 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 non-causal, 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 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, non-causal 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 non-causal 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 non-causal 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.

Recently, we have 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 highly structurally dynamic systems (i.e., 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 non-causal 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. Dependent types have been adopted as the tool for expressing these static guarantees. However, we believe that more practical type systems (such as system F) could be conservatively extended to make FHM safer without compromising their usability.

Further reading

The implementation of Hydra is available from <http://www.cs.nott.ac.uk/~ggg/> under the open source BSD license.

6.5.2 Elerea

Report by:	Patai Gergely
Status:	experimental, active

Elerea (Eventless reactivity) is a tiny continuous-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. A unique feature of the library is that cyclic dependencies are detected on the fly and resolved by inserting delays dynamically, unless the user does it explicitly.

As an example, the following code snippet is a possible way to define an approximation of our beloved trig functions:

```
(sine, cosine) <- mdo
  s <- integral 0 c
  c <- integral 1 (-s)
  return (s, c)
```

The library is minimal by design, and it provides low-level primitives one can build a cleaner set of combinators upon. Also, it is relatively easy to adapt it

to any imperative framework, although it is probably not a good choice to program primarily event-driven systems, because it is pull-based.

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.

Further reading

- <http://hackage.haskell.org/cgi-bin/hackage-scripts/package/elerea>
- <http://hackage.haskell.org/cgi-bin/hackage-scripts/package/elerea-examples>

6.6 Audio and Graphics

6.6.1 Audio signal processing

Report by:	Henning Thielemann
Status:	experimental, active development

In this project, audio signals are processed using pure Haskell code and the Numeric Prelude framework (→ 5.5.3). The highlights are:

- a basic signal synthesis backend for Haskore (→ 5.11.1),
- support for physical units while maintaining efficiency,
- frameworks for abstraction from sample rate, that is, the sampling rate can be omitted in most parts of a signal processing expression. We tried hard to preserve the functional style of programming and do not need Arrows and according notation.
- 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.
- 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.

Recent advances are:

- o Lazy time values to be used for gate signals,
- o enhanced type class framework for unifying lazy time values and signals expressed by lists, storable vectors or signal generators.
- o Connection toalsa bindings, in order to provide real-time sound synthesis controlled by MIDI events from keyboards or sequencers,
- o Stand-alone binding to Sox for audio format conversion and playback,
- o A pyramid filter for efficient computation of moving average and moving maximum for baseline detection of mass spectra,

Further reading

- o <http://www.haskell.org/haskellwiki/Synthesizer>
- o http://dafx04.na.infn.it/WebProc/Proc/P_201.pdf

6.6.2 easyVision

Report by:	Alberto Ruiz
Status:	experimental, active development

The *easyVision* project is a collection of experimental libraries for computer vision and image processing. The low level computations are internally implemented by optimized libraries (IPP, HOpenGL, hmatrix (→ 5.3.1), etc.). Once appropriate geometric primitives have been extracted by the image processing wrappers we can define interesting computations using elegant functional constructions. Future work includes cabalization of the main modules.

Further reading

<http://www.easyVision.googlepages.com>

6.6.3 photoname

Report by:	Dino Morelli
Status:	stable, maintained

See: <http://haskell.org/communities/05-2009/html/report.html#sect6.6.4>.

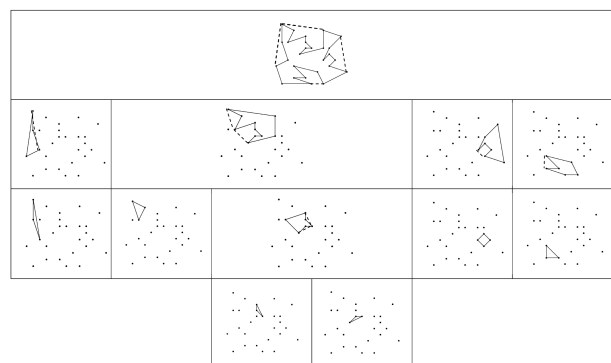
6.6.4 n-Dimensional Convex Decomposition of Polytops

Report by:	Farid Karimipour
Participants:	Rizwan Bulbul, Andrew U. Frank
Status:	active development

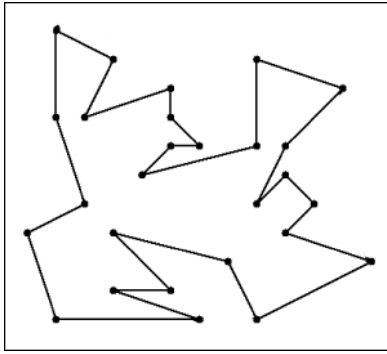
This is the continuation of the work on “Simplex-based Spatial Operations” (<http://haskell.org/communities/05-2009/html/report.html#sect6.6.5>), where we showed how to implement dimension independent spatial operations using the concept of n-simplexes. The results for

n-dimensional convex hull computation were demonstrated through a graphical user interface written with wxHaskell functions. In this report, we have applied the same approach to a more complicated spatial analysis, i.e., convex decomposition of polytops. Convexity is a simple but useful concept. Convex objects are much easier to deal with in terms of storage and operations: the intersection of two convex objects is a convex object, the calculation of areas/volumes is straightforward and so is the “point-in-polygon” problem, etc.

There are several approaches to decompose a polytop to convex components, some of which may be adapted for different dimensions. However, they still require separate implementations for each dimension. The main reason is lack of suitable data structures in the current programming languages. We use the n-simplexes to implement an n-dimensional algorithm for convex decomposition of polytops. It builds a tree of signed convex components: components in even levels are additive, whereas components in odd levels are subtractive. An example figure:



The algorithm starts with placing the convex hull of the input polytop as the root of the tree. Subtraction of the computed convex hull from the input polytop yields a set of split polytops, which are the elements of the next level of the tree. The procedure applies to the non-convex elements and it repeats until all of the elements are convex. The list data structures and list operations are used for implementation of the algorithm. Polytops are represented as a list of n-simplexes, which are described as a list of points, per se. It allows us to describe the operations for convex decomposition of polytops as a combination of operations of n-simplexes, which turns to be operations on lists. Since the representation and operations are defined independent of dimension, the decomposition algorithm can be used for polytops of any dimension. The following figure shows a 2D example polytop which is decomposed to convex components as the above figure.



Further reading

<http://haskell.org/communities/05-2009/html/report.html#sect6.6.5>

6.6.5 DVD2473

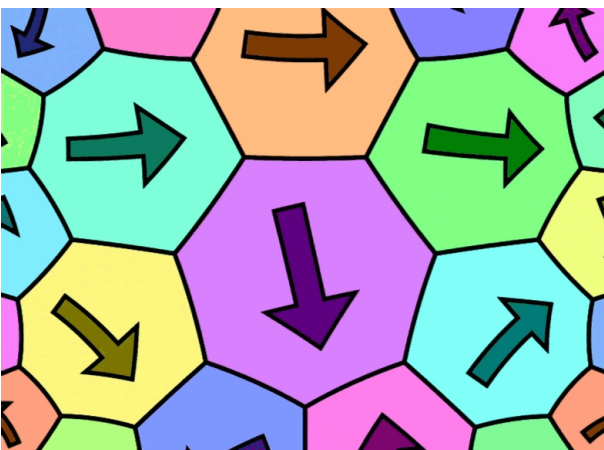
Report by:	Claude Heiland-Allen
Status:	complete

DVD2473 is a generative DVD video artwork.

The somewhat abstract title is a semi-literal description with all the descriptive elements taken out - it is a DVD of a perfect coloring in 24 colors of a 7,3 hyperbolic tiling. It uses the DVD virtual machine to navigate around the space at pseudo-random.

The mathematics (permutations and Moebius transforms) of the space were implemented in Haskell, with rendering with Gtk2Hs/Cairo including a little C code to copy and convert pixels from Cairo to ByteString. The video encoding used HSH to pipe ByteString to external tools. The XML control file for DVD authoring was generated with a separate Haskell program. A bash script automates building an iso from the source code directory.

There are no plans for further development, the artwork is finished.



Further reading

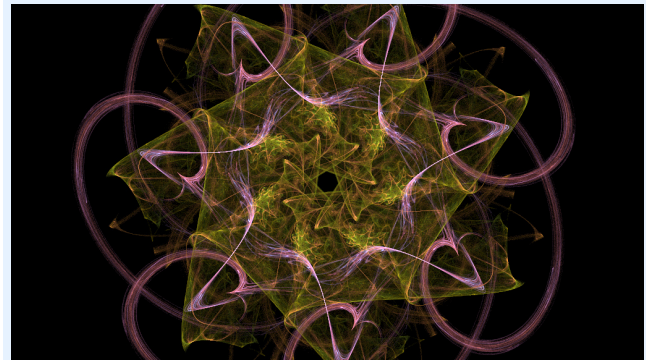
http://claudiusmaximus.goto10.org/cm/2009-01-07_dvd2473.html

6.6.6 F14m6e

Report by:	Claude Heiland-Allen
Status:	unstable

F14m6e is a fast fractal flame renderer, currently under active development. Inspired by flam3 and electric sheep, F14m6e performs all the rendering calculations on the GPU, using OpenGL shaders. The features so far include runtime generation of low level GLSL source code from abstract scene descriptions, smooth transitions between scenes using rigid transformations, and interactive control of animation and quality settings.

Contrasting to flam3, the aim is not exact image reproducibility, but to get fast enough that pre-rendering videos is not necessary — then a peer-to-peer network exchanging small scene descriptions would supercede a centralized file server and corresponding large bandwidth requirements. However, there is a long way to go before the electric sheep have anything to worry about.



Further reading

- o http://claudiusmaximus.goto10.org/cm/2009-08-28_fl4m6e_proof_of_concept.html
- o http://claudiusmaximus.goto10.org/cm/2009-09-24_fl4m6e_in_haskell.html
- o <http://claudiusmaximus.goto10.org/g/fl4m6e/examples/>
- o <http://flam3.com/>
- o <http://electricsheep.org/>

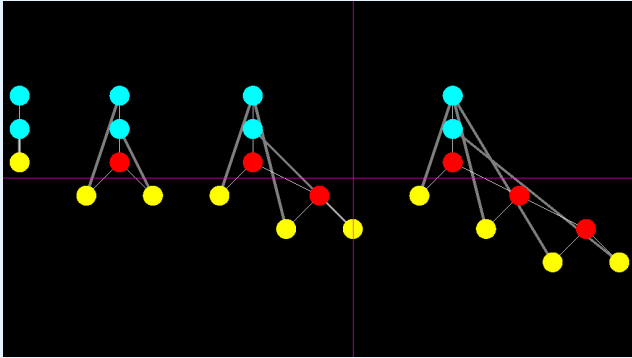
6.6.7 GULCI

Report by:	Claude Heiland-Allen
Status:	unstable

GULCI is a graphical untyped lambda calculus interpreter. Programs are written with mouse clicks and drags, and executed with a keypress. During execution the graph reduction is visualized. GULCI also dumps data on its standard output stream, suitable for sonification. The eventual intent is to use it for a short abstract code performance sometime in the future.

GULCI is the interactive descendant of the non-interactive (but also graphical) ULCiv1, which took the form of audiovisualisations of some simple arith-

metrical computations in untyped lambda calculus.



Further reading

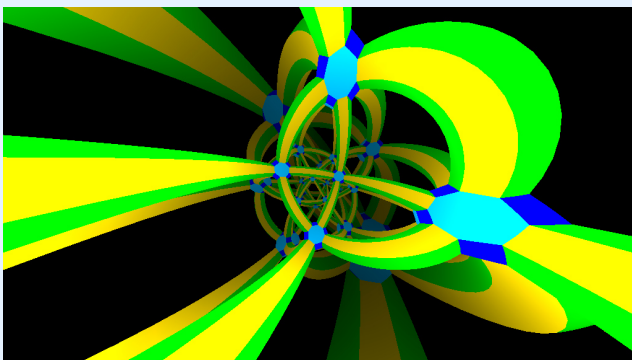
http://claudiusmaximus.goto10.org/cm/2009-06-19_untyped_lambda_calculus_interpretations_v1.html

6.6.8 Reflex

Report by:	Claude Heiland-Allen
Status:	experimental

Reflex is a program to interactively experience variously truncated regular 4D polytopes. The name is inspired by reflection symmetry groups and the eventual intent of using it as an interactive live audiovisual performance environment with quick reactions required, the idea for the software itself is inspired by The Symmetriad.

Starting from the Schläfli symbol p, q, r , Reflex constructs a 4D symmetry group, from which a wide variety of different forms can be visualized. The projection method results in aesthetically pleasing curves, enhanced by the animation. Currently Reflex accepts commands on its standard input, but the goal is to use a game controller to navigate through the world.



Further reading

- http://claudiusmaximus.goto10.org/cm/2009-10-15_reflex_preview.html
- <http://web.mit.edu/~axch/www/Symmetriad/index.html>
- http://en.wikipedia.org/wiki/Schläfli_symbol

6.7 Proof Assistants and Reasoning

6.7.1 Calculator

Report by:	Paulo Silva
Status:	unstable, work in progress

See: <http://haskell.org/communities/05-2009/html/report.html#sect6.7.1>.

6.7.2 Saoithín: a 2nd-order proof assistant

Report by:	Andrew Butterfield
Status:	ongoing

Saoithín (pronounced “Swee-heen”) is a GUI-based 2nd-order predicate logic proof assistant. The motivation for its development is the author’s need for support in doing proofs within the so-called “Unifying Theories of Programming” paradigm (UTP). This requires support for 2nd-order logic, equational reasoning, and meets a desire to avoid re-encoding the theorems into some different logical form. It also provides proof transcripts whose style makes it easier to check their correctness.

Saoithín is implemented in GHC 6.4 and wxHaskell 0.9.4, with elements of Mark Utting’s jaza tool for Z, and has been tested on a range of Windows platforms (98/XP/Vista), and should work in principle on Linux/Mac OS X.

A version of the software has been trialled out on 3rd-year students taking a Formal Methods elective course (<https://www.cs.tcd.ie/Andrew.Butterfield/Teaching/3BA31/#Software>) A first public release of the software under GPL will now happen during Summer 2009 — For now, Windows executables can be downloaded from the above link.

Further reading

<https://www.cs.tcd.ie/Andrew.Butterfield/Saoithin>

6.7.3 Inference Services for Hybrid Logics

Report by:	Guillaume Hoffmann
Participants:	Carlos Areces, Daniel Gorin

“Hybrid Logic” is a loose term covering a number of logical systems living somewhere between modal and classical logic. For more information on this languages, see <http://hylo.loria.fr>

The Talaris group at Loria, Nancy, France (<http://talaris.loria.fr>) and the GLyC group at the Computer Science Department of the University of Buenos Aires, Argentina (<http://www.glyc.dc.uba.ar/>) are developing a suite of tools for automated reasoning for hybrid logics, available at <http://code.google.com/p/intohylo/>. Most of them are (successfully) written in Haskell. See HyLoRes (<http://haskell.org/communities/05-2009/html/report.html#sect6.7.5>), HTab (→ 6.7.4),

and HGen (<http://haskell.org/communities/05-2009/html/report.html#sect6.7.7>).

6.7.4 HTab

Report by:	Guillaume Hoffmann
Participants:	Carlos Areces, Daniel Gorin
Status:	active development
Current release:	1.5.1

HTab is an automated theorem prover for hybrid logics (\rightarrow 6.7.3) based on a tableau calculus. It handles hybrid logic with nominals, satisfaction operators, converse modalities, universal and difference modalities, the down-arrow binder and role inclusion.

The source code is distributed under the terms of the GNU GPL.

Further reading

- Hoffmann, G. and Areces, C. *HTab: a terminating tableaux system for hybrid logic*. In *Methods for Modalities 5*, Cachan, France, 2007.
- Site and source: <http://code.google.com/p/intohylo/>

6.7.5 Sparkle

Report by:	Maarten de Mol
Participants:	Marko van Eekelen, Rinus Plasmeijer
Status:	stable, maintained

See: <http://haskell.org/communities/05-2009/html/report.html#sect6.7.8>.

6.7.6 Haskellelle

Report by:	Florian Haftmann
Status:	working

Since Haskell is a pure language, reasoning about equational semantics of Haskell programs is conceptually simple. To facilitate machine-aided verification of Haskell programs further, we have developed a converter from Haskell source files to Isabelle theory files: Haskellelle.

Isabelle itself is a generic proof assistant. It allows mathematical formulas to be expressed in a formal language and provides tools for proving those formulas in a logical calculus. One such formal language is higher-order logic, a typed logic close to functional programming languages. This is used as translation target of Haskellelle.

Both Haskellelle and Isabelle in combination allow to formally reason about Haskell programs, particularly verifying partial correctness.

The conversion employed by Haskellelle covers only a subset of Haskell, mainly since the higher-order logic

of Isabelle has a more restrictive type system than Haskell. A simple adaption mechanisms allows to tailor the conversion process to specific needs.

So far, Haskellelle is working, but there is little experience for its application in practice. Suggestions and feedback welcome. A tool demo is given at PEPM'10.

Further reading

<http://isabelle.in.tum.de/haskell.html> and <http://isabelle.in.tum.de/>

6.8 Modeling and Analysis

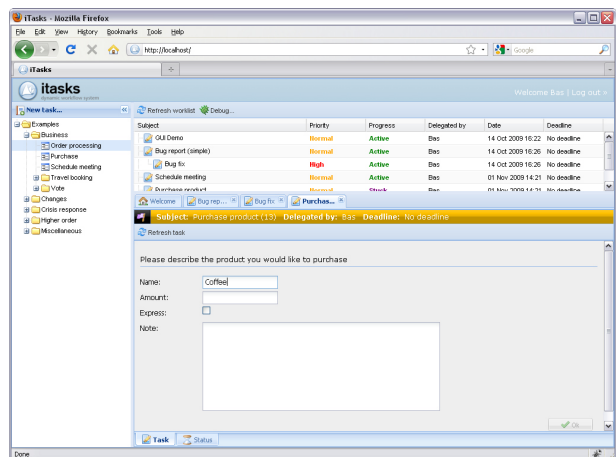
6.8.1 iTasks

Report by:	Bas Lijnse
Participants:	Rinus Plasmeijer, Peter Achten, Pieter Koopman, Thomas van Noort, Jan Martin Jansen, Erik Crombag
Status:	active development

The iTask system provides a set of combinators to specify workflow in the pure and functional language Clean (\rightarrow 3.2.3) at a very high level of abstraction. Workflow systems are automated systems in which tasks are coordinated that have to be executed by either humans or computers. Workflow specifications are supplemented with a generic foundation to generate executable multi-user workflow support systems that consist of a webservice-based server and a user-friendly Ajax client.

Compared to contemporary workflow systems, that often use simple graphical specification languages, the iTask system offers several advantages:

- Tasks are statically typed and can be higher-order.
- Combinators are fully compositional.
- Dynamic and recursive workflow is supported.
- Workflow instances can be modified during execution.



The iTask system makes extensive use of Clean's generic programming facilities for generating dynamic user-interfaces and data encoding/decoding.

Future plans

Currently, we are working on extending and stabilizing the iTask base system to a level where it can be used for serious applications. We are also exploring how even more dynamic and unpredictable workflows can be supported.

Further reading

- <http://itasks.cs.ru.nl/>
- <http://www.st.cs.ru.nl/Onderzoek/Publicaties/publicaties.html>

6.8.2 CSP-M animator and model checker

Report by:	Marc Fontaine
Status:	active development, download available

We develop an Haskell based, integrated CSP-M animator and model checker. CSP (Communicating-Sequential-Processes) is a formalism for concurrent systems, invented by Tony Hoare. CSP-M is the concrete syntax used by several CSP-tools.

Our Haskell-CSP-Tool features:

- FDR compatibility
- Fast computation of state spaces
- GTK+ based graphical user interface
- Support for shared-memory-parallelism / multicore CPUs

Binary releases are available for download via <http://www.stups.uni-duesseldorf.de/~fontaine/csp>.

The Parsec-based CSP-M parser of our project is also on Hackage. This is also the parser used by the ProB model-checker.

Further reading

<http://www.stups.uni-duesseldorf.de/~fontaine/csp>

6.9 Hardware Design

6.9.1 ForSyDe

Report by:	Ingo Sander
Participants:	Alfonso Acosta, Axel Jantsch, Jun Zhu
Status:	experimental

The ForSyDe (Formal System Design) methodology has been developed with the objective to move system-on-chip design to a higher level of abstraction. ForSyDe is implemented as a Haskell-embedded behavioral DSL.

The current released is ForSyDe 3.0, which includes a new deep-embedded DSL and embedded compiler with different backends (Simulation, Synthesizable VHDL and GraphML), as well as a new user-friendly tutorial.

The source code, together with example system models, is available from HackageDB under the BSD3 license.

Features

ForSyDe includes two DSL flavors which offer different features:

1. Deep-embedded DSL

Deep-embedded signals (`ForSyDe.Signal`), based on the same concepts as Lava (\rightarrow 8.9), are aware of the system structure. Based on that structural information ForSyDe's embedded compiler can perform different analysis and transformations.

- Thanks to Template Haskell, computations are expressed in Haskell, not needing to specifically design a DSL for that purpose
- Embedded compiler backends:
 - Simulation
 - VHDL (with support for Modelsim and Quartus II)
 - GraphML (with yFiles graphical markup support)
- Synchronous model of computation
- Support for components
- Support for fixed-sized vectors

2. Shallow-embedded DSL

Shallow-embedded signals (`ForSyDe.Shallow.Signal`) are modeled as streams of data isomorphic to lists. Systems built with them are restricted to simulation. However, shallow-embedded signals provide a rapid-prototyping framework which allows to simulate heterogeneous systems based on different models of computation (MoCs).

- Synchronous MoC
- Untimed MoC
- Continuous Time MoC
- Domain Interfaces allow connecting various subsystems with different timing (domains) regardless of their MoC

ForSyDe allows to integrate deep-embedded models into shallow-embedded ones. This makes it possible

to simulate a synthesizable deep-embedded model together with its environment, which may consist of analog, digital, and software parts. Once the functionality of the deep-embedded model is validated, it can be synthesized to hardware using the VHDL-backend of ForSyDe's embedded compiler.

Further reading

<http://www.ict.kth.se/forsyde/>

6.9.2 Kansas Lava

Report by:	Andy Gill
Participants:	Tristan Bull, Andy Gill, Garrin Kimmell, Erik Perrins, Ed Komp, Brett Werling
Status:	ongoing

Kansas Lava is an attempt to use the Lava design pattern with modern functional programming technology. In particular we scale up the ideas in Lava to operate on larger circuits and with large basic components.

- Kansas Lava uses a single principal **Signal** type for all types of signals. Some versions of Lava have overloaded `signal` to interpret signal in either a synthesis or simulation mode. Our experience is that a single concrete type is easier to work with in practice, and we have baked the two main interpretations into our **Signal** type. Ultimately this allows a closer fit between the specifications of behavior and synthesizable code.
- Like other Lava implementations before it, Kansas Lava supports both synthesis and simulation. The use case would typically be development of an executable model, refinement into a synthesizable variant, then further refinement for efficiency, and other considerations.
- Kansas Lava uses a modern FP style of Haskell. We allow **Signal** to be an applicative functor. Arithmetic is overloaded over `Signal`, so we can represent addition using `+`, multiplication using `*`, etc. Constant literals can also be `Signals`. This leads to a cleaner looking specifications.
- Kansas Lava has direct support for importing existing VHDL libraries as new, well typed primitives. This allows Kansas Lava to be used as a high-level glue between existing solutions.
- Kansas Lava includes a simple type checking over binary representations. Polymorphic specifications inside Lava will be instantiated to their specific, monomorphically sized implementation in VHDL, depending on the propagation of actual usage.
- In Haskell, requiring a 14-bit value is unusual, but in hardware, we often know and want to enforce a specific width. Kansas Lava uses an implementation of

sized types, built using type functions. This library, developed specifically for use with Kansas Lava, includes sized 1 and 2 dimensional matrixes, and sized signed and unsigned bit vectors.

We are writing Kansas Lava to address one problem (generate good hardware designs for communication circuits) and help facilitate a second (explore and understand correctness preserving optimizations of non-trivial hardware components). With telemetry circuits, the encodings and decoding mechanisms are typically expressed using matrix operations, so we pay careful attention to allowing clear encoding of such operations. In particular, we use type functions to clean up sized types, making for a cohesive addition to our Lava. A release is planned for early November, and will be available on Hackage.

Further reading

http://www.ittc.ku.edu/csdl/Kansas_Lava

6.10 Natural Language Processing

6.10.1 NLP

Report by:	Eric Kow
------------	----------

The Haskell Natural Language Processing community aims to make Haskell a more useful and more popular language for NLP. The community provides a mailing list, Wiki and hosting for source code repositories via the Haskell community server.

The Haskell NLP community was founded in March 2009. We are in the process of recruiting NLP researchers and users from the Haskell community. In the future, we hope to use the community to discuss libraries and bindings that would be most useful to us and ways of spreading awareness about Haskell in the NLP world.

Further reading

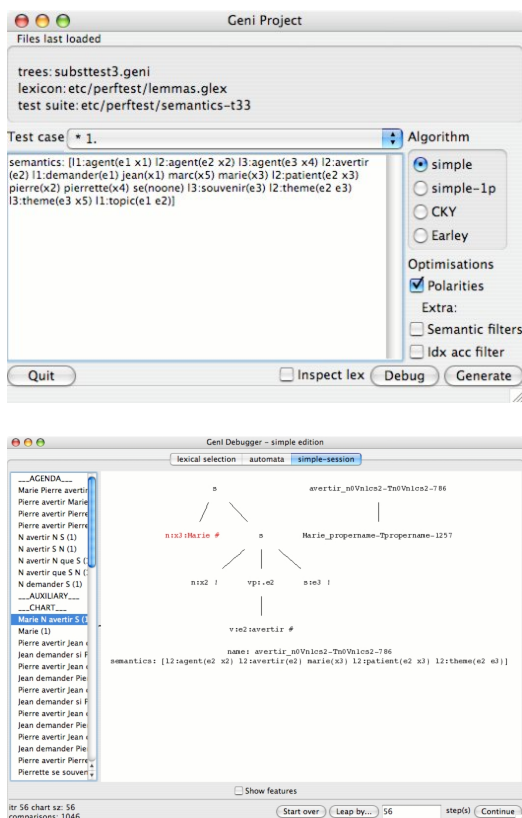
<http://projects.haskell.org/nlp>

6.10.2 GenI

Report by:	Eric Kow
------------	----------

GenI is a surface realizer for Tree Adjoining Grammars. Surface realization can be seen a subtask of natural language generation (producing natural language utterances, eg. English texts, out of abstract inputs). GenI in particular takes an FB-LTAG grammar and an input semantics (a conjunction of first order terms), and produces the set of sentences associated to 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.



GenI is available on Hackage, and can be installed via cabal-install. Our most recent release of GenI was version 0.20.1 (2009-10-01), which offers cleaner interactions with the third-party tools (using JSON), simpler installation on MacOS X and a user manual. 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>

6.10.3 Grammatical Framework

Report by: Krasimir Angelov
 Participants: Krasimir Angelov, Håkan Burden, Arne Ranta

Grammatical Framework (GF) is a programming language for multilingual grammar applications. It can be used as a more powerful alternative to Happy but in fact its main usage is to describe natural language grammars instead of programming languages. The language itself will look familiar for most Haskell or ML

users. It is a dependently typed functional language based on Per Martin-Löf's type theory.

An important objective in the language development was to make it possible to develop modular grammars. The language provides modular system inspired from ML but adapted to the specific requirements in GF. The modules system was exploited to a large extent in the Resource Libraries project. The library provides large linguistically motivated grammars for a number of languages. When the languages are closely related the common parts in the grammar could be shared using the modules system. Currently there are complete grammars for Bulgarian, Danish, English, Finnish, French, German, Interlingua, Italian, Norwegian, Russian, Spanish, and Swedish. There are work in progress grammars for Arabic, Catalan, Latin, Thai, and Hindi/Urdu. On top of these grammars a user with limited linguistic background can build application grammars for a particular domain.

On 24 June 2009 after one year of hard work the previous beta version of GF 3.0 is turned into stable release. This is a major refactoring of the existing system. The code base is about half in size and makes a clear separation between compiler and runtime system. A Haskell library is provided that allows GF grammars to be easily embedded in user applications. There is a translator that generates JavaScript code which allows the grammar to be used in web applications as well. The new release also provides new parser algorithm which works faster and is incremental. The incrementality allows the parser to be used for word prediction, i.e., someone could imagine a development environment where the programming language is natural language and the user still can press some key to see the list of words allowed in this position just like it is possible in Eclipse, JBuilder, etc.

We are continuing to work hard. Some of the projects that keeps us busy:

- Type checker for dependent types in the interpreter. Currently only the compiler has type checker.
- Semantic parsing — i.e., parser which considers not only the syntax but also the semantics of the language. The semantics is encoded using dependent types.
- New resource grammars for Romanian and Polish
- Visualisation of parse trees and dependency trees in addition to syntax trees
- Experiments with natural languages and big ontologies
- Building wide coverage Swedish grammar based on the resource grammar.

Further reading

www.digitalgrammars.com/gf

6.11 Others

6.11.1 IgorII

Report by:	Martin Hofmann
Participants:	Emanuel Kitzelmann, Ute Schmid
Status:	experimental, active development

IGORII is a new method and an implemented prototype for constructing recursive functional programs from a few non-recursive, possibly non-ground, example equations describing a subset of the input/output behavior of a target function to be implemented.

For a simple target function like `reverse` the sole input would be the following, the k smallest w.r.t. the input data type, examples:

```
reverse []      = []
reverse [a]     = [a]
reverse [a,b]   = [b,a]
reverse [a,b,c] = [c,b,a]
```

The result, shown below, computed by IGORII is a recursive definition of `reverse`, where the subfunctions `last` and `init` have been automatically invented by the program.

```
reverse []      = []
reverse (x:xs) = (last (x:xs)):(reverse (init (x:xs)))

last [x]        = x
last (x:y:ys)  = last (y:ys)
init [x]        = []
init (x:y:ys)  = x:(init (y:ys))
```

Recently, IGORII has been extended to use catamorphisms on lists as higher-order templates. After enabling the higher-order mode, given the previous examples of `reverse`, the system outputs the following solution:

```
reverse xs      = foldr snoc [] xs
snoc x xs       = foldr cons [x] xs
cons x (y:ys)   = x:(y:ys)
```

Features

- termination by construction
- handling arbitrary user-defined data types
- utilization of arbitrary background knowledge
- automatic invention of auxiliary functions as subprograms
- learning complex calling relationships (tree- and nested recursion)
- allowing for variables in the example equations
- simultaneous induction of mutually recursive target functions
- using list-catamorphisms as higher-order templates

Current Status and Future Plans

The original version of IGORII is implemented in the reflective rewriting based programming and specification language MAUDE. However, a Haskell implementation of the algorithm is the current research prototype. Both can be obtained from the project page.

A tool demo and a research paper about the use of catamorphisms as higher-order templates are presented at [PEPM 2010](#).

For the future, we plan to extend the higher-order context to use the catamorphism template for arbitrary algebraic data types. Furthermore, it is worth to investigate to which extent other morphisms can be incorporated.

Further reading

- <http://www.cogsys.wiai.uni-bamberg.de/effalip/>
- <http://www.inductive-programming.org/>

6.11.2 Roguestar

Report by:	Christopher Lane Hinson
Status:	early development

See: <http://haskell.org/communities/05-2009/html/report.html#sect6.12.2>.

6.11.3 LQPL — A quantum programming language compiler and emulator

Report by:	Brett G. Giles
Participants:	Dr. J.R.B. Cockett
Status:	v 0.8.3 experimental released

LQPL (Linear Quantum Programming Language) consists of a compiler for a functional quantum programming language and an associated assembler and emulator.

This programming language was inspired by Peter Selinger's paper "Toward a Quantum Programming Language". LQPL incorporates a simplified module/include system (more like C's include than Haskell's import), predefined unitary transforms, quantum control and classical control, algebraic data types, and operations on purely classical data. The compiler translates LQPL programs into an assembler language targeted to a quantum stack machine. The emulator, written using Gtk2Hs, translates the assembler to machine code and provides visualization of the program as it executes.

For example, the following procedure implements quantum teleportation:

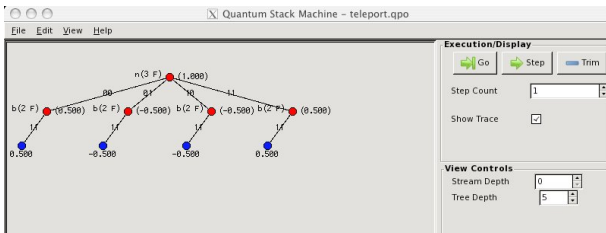
```
teleport :: (n:Qubit,a:Qubit,b:Qubit;b:Qubit) {
  Not a <= n ; Had n;
  measure a of
    |0> => {} |1> => {Not b};
  measure n of
```

```

}
|0> => {} |1> => {RhoZ b}
}

```

The emulator allows one to step through a program at the assembly level, displaying the quantum values as a tree. The figure below is a screen shot showing the results after the first measure in teleport.



Recent changes include a public release (available at the URL below), UI improvements and the ability load and compile LQPL source programs from the emulator.

Further reading

<http://pll.cpsc.ucalgary.ca/lqpl/index.html>

6.11.4 Yogurt

Report by: Martijn van Steenberg

Yogurt is a MUD client embedded in Haskell. The API allows users to define variables of arbitrary types and hooks that trigger on output from the MUD or input from the user. Other features include timers, multithreading, and logging. Most MUD clients rely on their own custom language; Yogurt, however, relies on Haskell. Even though Yogurt programs are full Haskell programs, Yogurt is able to dynamically load and reload them using the GHC API, effectively making Yogurt a scripting language.

Ideas for the future include compatibility with Tintin++ scripts to make migration to Yogurt even more tempting and an expect-like interface for easier interaction with processes.

Further reading

<http://code.google.com/p/yogurt-mud/>

6.11.5 Dyna 2

Report by: Wren Ng Thornton
 Participants: Nathaniel W. Filardo, Jason Eisner
 Status: active research

Dyna is a valued-logic programming language with first-class support for dynamic programming. A major goal of the language is to automate many of the common algorithms and optimizations used in natural language parsers and machine translation decoders, making them available for general logic programs.

Starting from Prolog we extend Horn clauses to "Horn equations" by associating each grounding of a

rule with a value (not just provability) and aggregating these values to get the value of an item. This extends logic programming with some elements of functional programming, including weighted-logic systems where the form of Horn equations is restricted to a semiring.

My master thesis work was developing a powerful type system which expresses algebraic data types with non-linearity constraints, refinement subtyping, and some aspects of dependent typing. This type system is further enhanced to allow heterogeneous storage of the same semantic type, so that different representations of "the same value" can be used simultaneously in different parts of a program.

Unlike most logic languages, Dyna will have a module system for separating proof universes, which allows multiple programs to share the same RTS instance and allows presenting programs, data sets, and deductive databases with the same API. Dyna will also support agenda-based mixed forward-/backward-chaining inference with memoization and declarative truth maintenance.

Previous work implemented a prototype compiler that generated C++ classes for a restricted form of the language. Currently we are implementing an interpreter in Haskell that covers a broader portion of the language, and are working on the formal underpinnings of these extensions. We intend to use this in the long term as a reference implementation for testing improved algorithms for a next generation Dyna compiler.

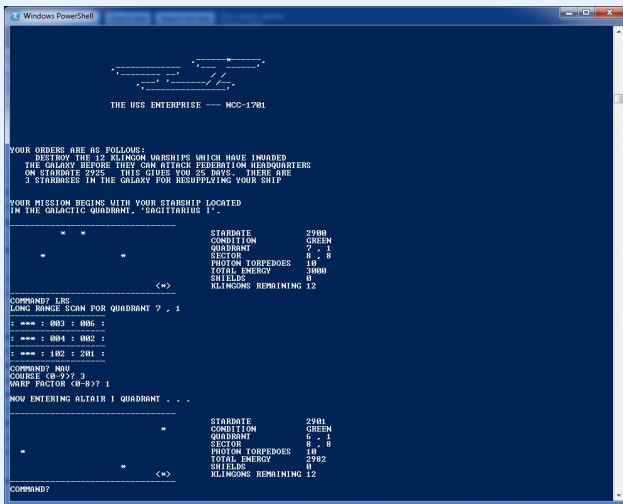
Further reading

- o Work on the type system and unification algorithms for Dyna 2 can be found in the following papers. These are currently not available online, though refined versions should be available soon.
 - W. Thornton (2008). "Typed Unification in Dyna: An Exploration of the Design Space." Masters Project Report, Johns Hopkins University.
 - W. Thornton (2008). "Heterogeneous Strategies for Unification: Variable-Value Ordering and Optimized Structures." Masters Paper, Johns Hopkins University.
- o This paper discusses program transformations for Dyna and gives an early view of the semantics for mixed inference: <http://www.cs.jhu.edu/~jason/papers/#fg06>.
- o This paper introduces Dyna 1 where Horn equations are restricted to a semiring: <http://cs.jhu.edu/~jason/papers/#emnlp05-dyna>.
- o In association with Dyna there is work on a graphical debugger, called Dynasty, which allows lazy exploration of hypergraphs (representing proof forests).
 - <http://cs.jhu.edu/~jason/papers/#infovis06>

6.11.6 Vintage BASIC

Report by: Lyle Kopnicky
Current release: 1.0.1
Portability: GHC 6.10

Vintage BASIC is an interpreter for microcomputer-era BASIC, written in Haskell. It is fully unit-tested, and implements all common features of the language. The web site includes games from Creative Computing's BASIC Computer Games, all of which can be run under the interpreter.



```
Windows PowerShell
THE USS ENTERPRISE --- NCC-1701

YOUR ORDERS ARE AS FOLLOWS:
DEFEAT THE 12 KLINGON WARSHIPS WHICH HAVE INVADED
THE GALAXY BEFORE THEY CAN ATTEMPT REPRODUCTION HEADQUARTERS
ON STARBATE 2725. THIS GIVES YOU 25 DAYS. THERE ARE
3 STARBATES IN THE GALAXY FOR RECAPTURING YOUR SHIP.

YOUR MISSION BEGINS WITH YOUR STARSHIP LOCATED
IN THE GALACTIC QUADRANT - SHOOTARBS 1.

   *      STARBATE 2988
   *      CONDITION GREEN
   *      QUADRANT 7 - 1
   *      SECTOR 8 - 8
   *      PHOTON TORPEDOS 18
   *      TOTAL ENERGY 3888
   *      SHIELDS 8
   *      KLINGONS REMAINING 12

COMMAND? INS
LONG RANGE SCAN FOR QUADRANT 7 , 1
: == : 083 : 086 :
: == : 084 : 082 :
: == : 102 : 201 :

COMMAND? NUU
COURSE (0-99) 3
WARP FACTOR (0-8) 1
NUU ENTERING ALTAR I QUADRANT . . .

   *      STARBATE 2981
   *      CONDITION GREEN
   *      QUADRANT 8 - 1
   *      SECTOR 8 - 8
   *      PHOTON TORPEDOS 18
   *      TOTAL ENERGY 2982
   *      SHIELDS 8
   *      KLINGONS REMAINING 12

COMMAND?
```

The interpreter makes use of a novel technique for implementing BASIC's dynamic control structures: resumable exceptions. For example, in BASIC loops, the FOR keyword becomes an exception handler, and the NEXT keyword throws an exception. Furthermore, the exceptions are caught in the continuation, rather than the containing expression. The handlers can also be selectively persisted after handling exceptions. Because of these two features, I refer to them as “durable traps”. The DurableTraps library is fully abstracted using monad transformers, and can be used in any program.

Further reading

<http://www.vintage-basic.net>

6.11.7 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 C compatibility layer, so the library can be used without modifications.

At the current state of the project only basic services are accessible from Haskell, i.e., you can load collision

shapes and step the simulation. More advanced Bullet features (constraints, soft body simulation etc.) will be added later.

Further reading

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

6.11.8 arbtt

Report by: Joachim Breitner
Status: working

The program arbtt, the automatic rule-based time tracker, allows you to investigate how you spend your time, without having to manually specify what you are doing. arbtt records what windows are open and active, and provides you with a powerful rule-based language to afterwards categorize your work. And it comes with documentation!

The program is very new and has found only a few users yet. The author is still looking for feedback, especially to rate its usefulness to decide whether it will be uploaded to Debian.

Further reading

- o <http://www.joachim-breitner.de/projects#arbtt>
- o <http://www.joachim-breitner.de/blog/archives/336-The-Automatic-Rule-Based-Time-Tracker.html>
- o http://darcs.nomeata.de/arbtt/doc/users_guide/

6.11.9 uacpid

Report by: Dino Morelli
Status: experimental, actively developed

uacpid is a daemon designed to be run in userspace that will monitor the local system's acpid socket for hardware events. These events can then be acted upon by handlers with access to the user's environment.

uacpid is available in binary form for Arch Linux through the AUR and can be acquired using darcs or other methods.

Further reading

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

7 Commercial Users

7.1 Well-Typed LLP

Report by:	Ian Lynagh
Participants:	Duncan Coutts

Well-Typed is a Haskell services company. We provide commercial support for Haskell as a development platform. We also offer consulting services, contracting, and training. For more information, please take a look at our website or drop us an e-mail at info@well-typed.com.

This has been a busy 6 months for us. As well as proprietary work, we have been pleased to have again been able to work on the GHC 6.12 release, as well as other core parts of the community infrastructure, such as Cabal and the Haskell Platform.

We have also been involved with the setting up and running of the Industrial Haskell Group (→ 7.8). We are pleased with the work the IHG have done in its first 6 months, and have high hopes for the future.

Overall, it has been a good 6 months for us, and we are looking forward to the challenges of the next 6. We have some exciting stuff coming up, and will be looking into expansion.

Further reading

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

7.2 Credit Suisse Global Modeling and Analytics Group

Report by:	Ganesh Sittampalam
------------	--------------------

GMAG, the quantitative modeling group at Credit Suisse, has been using Haskell for various projects since the beginning of 2006, with the twin aims of improving the productivity of modelers and making it easier for other people within the bank to use GMAG models.

Many of GMAG's models use Excel combined with C++ addin code to carry out complex numerical computations and to manipulate data structures. This combination allows modelers to combine the flexibility of Excel with the performance of compiled code, but there are significant drawbacks: Excel does not support higher-order functions and has a rather limited and idiosyncratic type system. It is also extremely difficult to make reusable components out of spreadsheets or subject them to meaningful version control.

Because Excel is (in the main) a side-effect free environment, functional programming is in many ways a

natural fit, and we have been using Haskell in various ways to replace or augment the spreadsheet environment.

Our past projects include:

- Adding higher-order functions to Excel, implemented via (Haskell) addin code.
- Tools to transform spreadsheets into directly executable code.
- A “lint” tool to check for common errors in spreadsheets.

Our main project for the last couple of years has been Paradise, a domain-specific language embedded in Haskell for implementing reusable components that can be compiled into multiple target forms. Current backends are Excel spreadsheets and .NET components based on either Winforms or WPF.

A number of modelers have been exposed directly to Haskell by using Paradise, and they have generally picked it up fairly quickly. All new recruits are introduced to Haskell as part of our internal training program.

Our main focus at the moment is the automatic generation of Paradise models for a particular financial product, starting from an algebraic datatype that defines the product. Modelers can override parts of the automatically generated model with a hand-crafted Paradise component if they choose to, providing a good trade-off between speed of development and “beautiful” results.

Further reading

- CUFP 2006 talk about Credit Suisse:
<http://cufp.galois.com/slides/2006/HowardMansell.pdf>
- ICFP 2008 experience report about Paradise:
<http://www.earth.li/~ganesh/research/paradise-icfp08/paper.pdf>
<http://www.earth.li/~ganesh/research/paradise-icfp08/talk.pdf>

7.3 Bluespec tools for design of complex chips

Report by:	Rishiyur Nikhil
Status:	commercial product

Bluespec, Inc. provides a language, BSV, which is being used for all aspects of ASIC and FPGA system 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 universal, accommodating the diverse range of blocks found in SoCs, 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 RTL (Verilog), which can be further synthesized into netlists for ASICs and FPGAs using other commercial tools. Automatic synthesis from atomic transactions enables design-by-refinement, where an initial executable approximate design is systematically transformed into a quality implementation by successively adding functionality and architectural detail. Other products include fast BSV simulation and development tools. Bluespec also uses Haskell to implement its tools (well over 100K lines of Haskell).

This industrial strength tool has enabled some large designs (over a million gates) and significant architecture research projects in academia and industry. This kind of research was previously feasible only in software simulation. BSV permits the same convenience of expression as SW languages, and its synthesizability further allows execution on FPGA platforms at three orders of magnitude greater speeds, making it possible now to study realistic scenarios.

Status and availability

BSV tools, available since 2004, are in use by several major semiconductor companies and universities. The tools are free for academic teaching and research.

Recent news (last 6 months): (1) Much new infrastructure and libraries to move computation kernels easily onto commodity FPGA boards, for greater speed and/or lower energy; (2) a strange loop, where one customer is applying the capability 1 to a computation kernel written in Haskell; and (3) development of PAClib (Pipeline Architecture Combinators) that make extensive use of higher-order functions to describe DSP algorithms succinctly and with powerful architectural parameterization, exceeding the capabilities of tools that synthesize hardware from C codes.

Further reading

- R.S.Nikhil, *Bluespec, a General-Purpose Approach to High-Level Synthesis Based on Parallel Atomic Transactions*, in *High Level Synthesis: from Algorithm to Digital Circuit*, Philippe Coussy and Adam Morawiec (editors), Springer, 2008, pp. 129-146.
- Small illustrative examples: <http://www.bluespec.com/wiki/SmallExamples>
- Winning entry in MEMOCODE 2008 design contest: <http://rijndael.ece.vt.edu/memocontest08/>

- MIT courseware, “Complex Digital Systems”: <http://csg.csail.mit.edu/6.375>
- A fun example with many functional-programming features — BluDACu, a parameterized Bluespec hardware implementation of Sudoku: <http://www.bluespec.com/products/BluDACu.htm>

7.4 Galois, Inc.

Report by:

Andy Adams-Moran

Galois is an employee-owned software development company based in Beaverton, Oregon, U.S.A. Galois started in late 1999 with the stated purpose of using functional languages to solve industrial problems. These days, we emphasize the needs of our clients and their problem domains over the techniques, and the slogan of the Commercial Users of Functional Programming Workshop (see <http://cufp.functionalprogramming.com/>) exemplifies our approach: Functional programming as a *means*, not an *end*.

Galois develops software under contract, and every project (bar three) that we have ever done has used Haskell. The exceptions used ACL2, Poly-ML, SML-NJ, and OCaml, respectively, so functional programming languages and formal methods are clearly our “secret sauce”. We deliver applications and tools to clients in industry and the U.S. government. Some diverse examples: Cryptol, a domain-specific language for cryptography (with an interpreter and a compiler, with multiple targets, including FPGAs); a GUI debugger for a specialized microprocessor; a specialized, high assurance, cross-domain web and file server, and Wiki for use in secure environments, and numerous smaller research projects that focus on taking cutting-edge ideas from the programming language and formal methods community and applying them to real world problems.

Web-based technologies are increasingly important to our clients, and we believe Haskell has a key role to play in the production of reliable, secure web software. The culture of correctness Haskell encourages is ideally suited to web programming, where issues of security, authentication, privacy, and protection of resources abound. In particular, Haskell’s type system makes possible strong static guarantees about access to resources, critical to building reliable web applications.

To help push further the adoption of Haskell in the domain of web programming, Galois released a suite of Haskell libraries, including:

- json: Support for JavaScript Object Notation
- xml: A simple, lightweight XML parser/generator.
- utf8-string: A UTF8 layer for IO and Strings.
- selenium: Communicate with a Selenium Remote Control server.

- `curl`: `libcurl` is a rich client-side URL transfer library.
- `sqlite`: Haskell binding to `sqlite3` databases.
- `feed`: Interfacing with RSS and Atom feeds
- `mime`: Haskell support for working with MIME types.

Continuing our deep involvement in the Haskell community, Galois was happy to sponsor the two Haskell hackathons held in the past year, Hac 07 II, in Freiburg, Germany, and Hac4 in Gothenburg, Sweden. Galois also sponsored the second BarCamp Portland, held in early May 2008.

Further reading

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

7.5 IVU Traffic Technologies AG Rostering Group

Report by:	Michael Marte
Status:	released

The roosting group at IVU Traffic Technologies AG has been using Haskell to check rosters for compliance with the “EC Regulation No 561/2006 on the harmonization of certain social legislation relating to road transport” which “lays down rules on driving times, breaks and rest periods for drivers engaged in the carriage of goods and passengers by road”. Due to combinatorial rest-time compensation rules, EC 561/2006 instances are search problems and at times pretty hard to solve.

Our implementation is based on an embedded DSL to combine the regulation’s single rules into a solver that not only decides on instances but, in the case of a faulty roster, finds an interpretation of the roster that is “favorable” in the sense that the error messages it entails are “helpful” in leading the dispatcher to the resolution of the issue at hand.

Our EC 561/2006 solver comprises about 1700 lines of Haskell code (including about 250 lines for the C API), is compiled to a DLL with `ghc`, and linked dynamically into C++ and Java applications. The solver is both reliable (due to strong static typing and referential transparency — we have not experienced a failure in three years) and efficient (due to constraint propagation, a custom search strategy, and lazy evaluation).

Our EC 561/2006 component is part of the IVU.crew software suite and as such is in wide-spread use all over Europe, both in planning and dispatch. So the next time you enter a regional bus, chances are that the driver’s roster was checked by Haskell.

Further reading

- [EC 561/2006 at EurLex](#)
- [The IVU.suite for public transport](#)

7.6 Tupil

Report by:	Chris Eidhof
Participants:	Eelco Lempsink

(,) tupil

Tupil builds reliable web software with Haskell. Using Haskell’s powerful ways of abstraction, we feel we can develop even faster than with dynamic scripting languages but with the safety and performance of a language that is statically checked and compiled.

In the last year we were able to successfully use Haskell for different projects: high score web services, music mashups, a payment system for a client and more. It would not have been possible without the vast amount of packages that are available for Haskell these days.

Further reading

- <http://tupil.com>
- <http://blog.tupil.com>

7.7 Aflexi Content Delivery Network (CDN)

Report by:	Kim-Ee Yeoh
------------	-------------

The Aflexi Content Delivery Network (CDN) is a federated solution to cost-effective content delivery for publishers, value-added capacity right-sizing for web-hosting providers, and a more responsive Internet experience for end-users.

Key elements of the Aflexi CDN comprise a server-side software package that webhosting providers can license to CDN-enable their servers and hosted websites, and a marketplace where a provider can federate with other providers to expand its CDN footprint. Our motto is “Unifying Capacity.”

At the heart of the platform is the ability for publishers to transparently combine Aflexi-enabled providers and migrate among a publisher-selected subset. Competition between providers ensures that publishers get a market-efficient rate for delivery bandwidth by making informed decisions based on platform metrics of providers’ Quality of Service. By trading excess capacity with other providers, they in turn benefit from the disruptive innovation in capacity recalibration and additional revenue streams afforded by the Aflexi CDN platform.

Aflexi uses Haskell for critical components of its back-end infrastructure. Haskell allows us to rapidly prototype our software efforts via its rich store of open-source libraries on Hackage. Supported by a set of composable concurrency abstractions built on fast

lightweight threads, our Haskell code sports more resilient fail-safe features and higher performance while at the same time employing fewer lines of code that ultimately translate to fewer bugs.

Other Haskell projects in development include a domain-specific language (DSL) with termination guarantees (à la *Total Functional Programming*). The DSL furnishes a framework for describing the policies governing content redirection.

Status and availability

The Aflexi CDN platform pre-launched at the start of 2009.

Further reading

<http://aflexi.net/>

7.8 Industrial Haskell Group

Report by: Ian Lynagh

The Industrial Haskell Group (IHG) is an organization to support the needs of commercial users of Haskell. It was formed in early 2009, and has already made a significant contribution to the up-coming GHC 6.12 release: In the first 6 months collaborative development scheme, the IHG has funded work on dynamic libraries, more flexible Integer library support for GHC, and Cabal development work. The details of these projects are on the website.

Following the success of this scheme, we will be running another 6 month scheme, with the work beginning in January. Interested companies should check out the website for details.

We have also added “associate membership” and “academic membership” options, for companies and academic groups that wish to support Haskell development without the larger commitment necessary for the collaborative development scheme. Again, details are 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/>

7.9 typLAB

Report by: Sebastiaan Visser
Participants: Lon Boonen, Erik Hesselink, Salar al Khafaji



TypLAB is a startup company located in the city center of Amsterdam. We investigate and develop new ways of creating and consuming online content. Our current focus is in building an online environment in which users can manage content in unconventional ways.

The Happstack powered server application, the automated deployment scripts, the JavaScript preprocessors; all code running at the server is written entirely in Haskell. The vast amount of Haskell packages, especially for XML manipulation and generic programming, allow us to easily interface with our Berkeley XML database back-end. A large part of our application is written in JavaScript and runs in the client. Most of the JavaScript code is heavily inspired by functional (reactive) programming and enables us to achieve a very high level of abstraction, even in the web browser.

TypLAB is showing that combining the theoretical foundations of computer science with the day-to-day practice of the web allows for building high-quality web-applications. Still a lot of work has to be done before our first beta will see the light. Please keep in touch with our progress by checking out our weblog.

Further reading

- o <http://typlab.com>
- o <http://blog.typlab.com>

8 Research and User Groups

8.1 Functional Programming Lab at the University of Nottingham

Report by: Liyang HU

The School of Computer Science at the University of Nottingham has recently formed the *Functional Programming Laboratory*, a new research group focused on all theoretical and practical aspects of functional programming, together with related topics such as type theory, category theory, and quantum programming.

The laboratory is led jointly by Thorsten Altenkirch and Graham Hutton, with Henrik Nilsson and Venanzio Capretta as academic staff. With 4 more research staff and some 10 PhD students in our group, we have a wide spectrum of interests:

Containers

Nottingham has been home to the EPSRC grant on *containers*, a semantic model of functional data structures. Thorsten Altenkirch, Peter Hancock, Peter Morris, and Rawle Prince are working with containers to both write and reason about programs. Peter Morris has recently finished his PhD, which used containers as a basis for generic programming with dependent types.

Dependently Typed Programming (DTP)

Peter Morris and Nicolas Oury are working on Epigram, while Nils Anders Danielsson is involved in the development of Agda (\rightarrow 3.2.2). Our interests lie both in the pragmatics of using DTP, as witnessed by work on libraries and tools, and in foundational theory, including the Observational Type Theory underlying Epigram 2 and James Chapman’s work on normalization. DTP is also used to control and reason about effects, and a number of us are using Agda as a proof assistant to verify programs or programming language theory.

Functional Reactive Programming (FRP)

The FRP team are working on FRP-like and FRP-inspired declarative, domain-specific languages. Under Henrik Nilsson’s supervision, Neil Sculthorpe is working on a new, scalable FRP language based on reactive components with multiple inputs and outputs, while George Giorgidze is applying the advantages of FRP to non-causal modeling with the aim of designing a new, more expressive and flexible language for non-causal, hybrid modeling and simulation (\rightarrow 6.5.1). Tom Nielsen is implementing a declarative language for experiments, simulations, and analysis in neuroscience.

A theme that permeates our work is implementation through embedding in typed functional languages such as Haskell or Agda (\rightarrow 3.2.2). The team also maintains Yampa, the latest Haskell-based implementation of FRP.

Quantum Programming

Thorsten Altenkirch and Alexander S Green have been working on the Quantum IO Monad (QIO), an interface from Haskell to Quantum Programming. Taking advantage of abstractions available in Haskell we can provide QIO implementations of many well-known quantum algorithms, including Shor’s factorization algorithm. The implementation also provides a constructive semantics of quantum programming in the form of a simulator for such QIO computations.

Reasoning About Effects

Graham Hutton and Andy Gill recently formalized the worker/wrapper transformation for improving the performance of functional programs. Wouter Swierstra and Thorsten Altenkirch have produced functional specifications of the IO monad, as described in Wouter’s forthcoming PhD thesis. Mauro Jaskelioff developed a new monad transformer library for Haskell, which provides a uniform approach to lifting operations. Diana Fulger and Graham Hutton are investigating equational reasoning about various forms of effectful programs. Liyang HU and Graham Hutton are working on verifying the correctness of compilers for concurrent functional languages, including a model implementation of software transactional memory.

Teaching

Haskell plays an important role in the undergraduate program at Nottingham, as well as our China and Malaysia campuses. Modules on offer include Functional Programming, Advanced FP, Mathematics for CS, Foundations of Programming, Compilers, and Computer-Aided Formal Verification, among others.

Events

The FP Lab plays a leading role in the Midlands Graduate School in the Foundations of Computing Science, the British Colloquium for Theoretical Computer Science, and the Fun in the Afternoon seminar series on functional programming.

FP Lunch

Every Friday, we gather for lunch with helpings of informal, impromptu-style whiteboard discussions on recent developments, problems, or projects. Summaries of our weekly meetings can be found on the frequently cited *FP Lunch blog*, giving a lively picture of ongoing research at Nottingham.

Later in the afternoon, there is usually a formal hour-long seminar. We are always keen on speakers in any related areas: do get in touch with [Thorsten Altenkirch](mailto:Thorsten.Aalten@cs.nott.ac.uk) (txa@cs.nott.ac.uk) if you would like to visit. See you there!

8.2 Artificial Intelligence and Software Technology at Goethe-University Frankfurt

Report by: David Sabel
Participants: Manfred Schmidt-Schauß

Deterministic calculi. We proved correctness of strictness analysis using abstract reduction. Our proof is based on the operational semantics of an extended call-by-need lambda calculus which models a core language of Haskell. Furthermore, we proved equivalence of the call-by-name and call-by-need semantics of an extended lambda calculus with `letrec`, `case`, and constructors. Recently, we extended the investigation of call-by-need `letrec` calculi to polymorphic typing and showed correctness of type dependent program transformations.

Nondeterministic calculi. We explored several nondeterministic extensions of call-by-need lambda calculi and their applications where an emphasis of our research lies in proving program equivalences w.r.t. contextual equivalence. In particular, we analyzed a model for a lazy functional language with direct-call I/O providing a semantics for `unsafePerformIO` in Haskell. We investigated a call-by-need lambda-calculus extended by parallel-or and its applications as a hardware description language. We analyzed a call-by-need lambda calculus extended with McCarthy's `amb` and an abstract machine for lazy evaluation of concurrent computations.

Simulation-based proof techniques. We have shown that mutual similarity is a sound proof method w.r.t. contextual equivalence in a class of untyped higher-order non-deterministic call-by-need lambda calculi. For call-by-need calculi with `letrec` we obtained two results: Most recently, we have shown that applicative bisimulation is correct in deterministic call-by-need lambda calculi with recursive `let`. For non-deterministic call-by-need calculi with `letrec` usual definitions of mutual similarity are unsound. In collaboration with *Elena Machkasova* we obtained correctness of a variation of finite simulation for proving con-

textual equivalence in an extended non-deterministic call-by-need lambda-calculus with `letrec`.

Concurrency primitives. We analyzed the expressivity of concurrency primitives in various functional languages. In collaboration with *Jan Schwinghammer* and *Joachim Niehren*, we showed how to encode Haskell's MVars into a lambda calculus with storage and futures which is an idealized core language of Alice ML. We proved correctness of the encoding using operational semantics and the notions of adequacy and full-abstractness of translations. In her final year thesis *Martina Willig* analyzed the encoding of other concurrency abstractions and implemented them in Concurrent Haskell.

Further reading

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

8.3 Functional Programming at the University of Kent

Report by: Olaf Chitil

The Functional Programming group at Kent is a subgroup of the newly formed 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. Recently Olaf Chitil and Doaitse Swierstra published a paper on simple and efficient implementations of pretty printing combinators. The development of Heat ([→ 4.3.2](#)), a deliberately simple IDE for teaching Haskell, made further progress over the summer. Neil Brown and Adam Sampson are developing an ocaml compiler in Haskell (`Tock`). They recently developed Alloy, a library for fast traversals and manipulations of tree-structured data.

Further reading

- FP group: <http://www.cs.kent.ac.uk/research/groups/tcs/fp/>
- Refactoring Functional Programs: <http://www.cs.kent.ac.uk/projects/refactor-fp/>
- Pretty Printing Combinators: S. Doaitse Swierstra and Olaf Chitil. *Linear, bounded, functional pretty-printing*. Journal of Functional Programming, 19(01):1-16, January 2009.
- Tracing and debugging with Hat: <http://www.haskell.org/hat>

- o Heat: <http://www.cs.kent.ac.uk/projects/heat/>
- o Tock: <http://projects.cs.kent.ac.uk/projects/tock/>
- o Alloy: Brown, N. C. and Sampson, A. T. 2009. *Alloy: fast generic transformations for Haskell*. In Proceedings of the 2nd ACM SIGPLAN Symposium on Haskell, Haskell '09 pages 105–116.

8.4 Foundations and Methods Group at Trinity College Dublin

Report by:	Andrew Butterfield
Participants:	Glenn Strong, Hugh Gibbons, Edsko de Vries

The Foundations and Methods Group focuses on formal methods, category theory, and functional programming as the obvious implementation method. A sub-group focuses on the use, semantics, and development of functional languages, covering such areas as:

- o Supporting OO-Design technique for functional programmers
- o Using functional programs as invariants in imperative programming
- o Developing a GUI-based 2nd-order equational theorem prover (<http://haskell.org/communities/05-2009/html/report.html#sect6.7.3>)
- o New approaches to uniqueness typing, applicable to Hindley-Milner style type-inferencing
- o Equational reasoning for Concurrent Haskell (new)

We have also managed to introduce a new elective course in functional programming at TCD which will be based on the “Real World Haskell” textbook.

Further reading

https://www.cs.tcd.ie/research_groups/fmg/

8.5 Formal Methods at DFKI Bremen and University of Bremen

Report by:	Christian Maeder
Participants:	Mihai Codescu, Dominik Lücke, Christoph Lüth, Christian Maeder, Till Mossakowski, Lutz Schröder

See: <http://haskell.org/communities/05-2009/html/report.html#sect8.5>.

8.6 Haskell at K.U.Leuven, Belgium

Report by:	Tom Schrijvers
Participants:	Pieter Wuille

We are a two-man unit of functional programming research within the Declarative Languages and Artificial Intelligence group at the Katholieke Universiteit Leuven, Belgium.

Our main project centers around the *Monadic Constraint Programming* (MCP) framework. An initial article on the MCP framework by Tom Schrijvers, Peter Stuckey and Phil Wadler is available. It explains how the framework captures the generic aspects of Constraint Programming in Haskell. Of particular interest is the solver-independent framework for compositional search strategies.

Currently we are extending the framework to act as a finite domain modeling language for both the problem description and the search component. The model in Haskell serves as a high-level front-end for a state-of-the-art Constraint Programming system such as Gecode (C++). Models can be compiled to C++ code, can be solved by calling Gecode from Haskell at runtime, or can be solved purely in Haskell itself.

Our other Haskell-related projects are:

- o *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.
- o *Type Checking*: Recent results are on type inference for GADTs, type invariants, and type checking for type families. Ongoing work concerns the simplification of type checking for Haskell extensive type system, and adding new extensions. This is joint work with Martin Sulzmann, Simon Peyton Jones, Manuel Chakravarty, Dimitrios Vytiniotis, Stefan Monnier, Louis-Julien Guillemette and Dominic Orchard.

Further reading

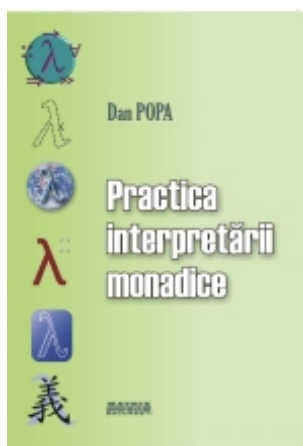
- o <http://hackage.haskell.org/cgi-bin/hackage-scripts/package/monadiccp>
- o <http://www.cs.kuleuven.be/~toms/Haskell/>
- o <https://www.cs.kuleuven.be/~pieterw/site/Research/Papers/>

8.7 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. We hope to pass the barrier of the 20000th click on the main page this month. The numbers of students and teachers interested in Haskell is increasing. Students begin to have projects using Haskell in order to pass the License Exams. But it is just a beginning. Interests in Data Base Programming with Haskell are growing. (A surprise!).

A book previously published by Mihai Gontineac was released as a free resource. A new book, “The Practice Of Monadic Interpretation” by Dan Popa has been published in November 2008.



The book has a nice foreword written by Simon P.J. and is sharing the experience of a year of interpreter building (2006). It is intended as a student’s and teacher’s guide to the practical approach of monadic interpretation. The book will probably be used during this academic year in 2–4 universities across Romania (in Iasi, Bacau, Cluj-Napoca).

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. Some teachers from a high school have requested training concerning Rodin. Rodin was asked to become a FOSS (Free & Open Source Software).

A group of researchers from the field of linguistics located at the State Univ. from Bacau (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.

At Bacau State University, we have teaching Haskell on both 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. “An Introduction to Haskell by Examples” had traveled to The Transilvania Univ. (Brasov) and we are expecting Haskell to be used there, too, during this academic year. Other libraries had received manuals and even donations (in books, of course). Editors seem to be interested by the Ro/Haskell movement, and some of them have already declared the intention of helping us by investing capital in the Haskell books production. A well known Publishing House (MatrixRom) asked us to be the Official Publishing House of the Ro/haskell Group.

Dan Popa is reporting a new technology in order to build the Modular Abstract Syntax Tree of a language processor without Maybe, without Catamorphisms and without Haskell extensions. A paper is available.

There are some unsolved problems, too: PhD. Advisors (specialized in monads, languages engineering, and Haskell) are almost impossible to find. This fact seems to block somehow the hiring of good specialists in Haskell. There was even a funny case when somebody hired to teach Haskell was tested and interviewed by a LISP teacher. Of course, the exam was more or less about lists.

Further reading

- o Ro/Haskell: <http://www.haskell.org/haskellwiki/Ro/Haskell>
- o Rodin: <http://www.haskell.org/haskellwiki/Rodin>

8.8 fp-syd: Functional Programming in Sydney, Australia.

Report by: Ben Lippmeier
Participants: Erik de Castro Lopo

We are a seminar and social group for people in Sydney, Australia interested in Functional Programming and related fields. We meet on the third Thursday of each month and regularly get 25–30 attendees, with a 70/30 industry/research split. Talks this year have included “Intro to PLT Scheme”, “A Haskell library for chart plotting”, and “Program extraction in a theorem prover like Coq (or Isabelle)”. 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>

8.9 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 Functional Programming research group has 5 faculty members and 9 postdoc and doctoral students. Research is going on in various exciting topics:

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. A draft 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.xhtml?pubid=99387>.

Natural language technology Grammatical Framework ([→ 6.10.3](#)) 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.

Generic Programming Starting with Polytypic Programming in 1995 there is a long history of generic programming research at Chalmers. Recent developments include work on dependent types (a JFP paper + library around “Algebra of Programming using Agda”), two survey papers “C++ Concepts =? Haskell Type Classes” and “Comparing GP Libs in Haskell” and applications to sustainable development with the Potsdam Institute for Climate Impact Research (<http://www.pik-potsdam.de/>). Patrik Jansson (with Sibylle Schupp) chaired the recent Workshop on Generic Programming.

Text editors Yi is a text editor in and for Haskell. It is a community project, but much development

and maintenance happens at Chalmers. For example, enhancing Yi was the goal of two master’s theses, and motivated research about incremental parsing: <http://publications.lib.chalmers.se/cpl/record/index.xhtml?pubid=94979>.

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 ([→ 3.2.2](#)) 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”.

DSP programming Feldspar 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. The motivating application is telecom processing, but the language is intended to be more general. As a first stage, we focus on the data-intensive numeric algorithms which are at the core of any DSP application, but in the future, we also plan to extend the language to deal with more system-level aspects. The data processing language is purely functional and highly inspired by Haskell. Currently the language is implemented as an embedded language in Haskell.

The implementation (including a code generator developed by ELTE University) will be publicly released within the coming weeks. Keep an eye on the project page: <http://sourceforge.net/projects/feldspar/>.

Hardware design/verification The functional programming group has developed three different hardware description languages — Lava, Wired and Chalk (chronological order) — implemented in Haskell. Each language targets a different abstraction level. The basic idea behind all three is to model circuits as functions from inputs to outputs. This allows structural hardware description in standard functional programming style.

Chalk is a new language for architecture design. Once you have defined a Chalk circuit, you can simulate it, or explore it further using non-standard interpretations. This is particularly useful if you want

to perform high-level power and performance analysis early on in the design process.

In **Lava**, circuits are described at the gate level (with some RTL support). The version developed at Chalmers (<http://www.cs.chalmers.se/~koen/Lava/>) has a particular aim to support formal verification in a convenient way. The version developed at Xilinx Inc. (<http://raintown.org/lava/>) focuses on FPGA core generation, and has been successfully used in real industrial design projects.

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. Some features of Wired are:

- Initial description can be purely functional (a la Lava).
- Incremental specification of physical aspects.
- Accurate, wire-aware timing/power analysis within the system.
- Support for an academic 45nm cell library.

Unfortunately, Wired is not actively developed at the moment, but the system has recently been used to explore the layout of multipliers (<http://publications.lib.chalmers.se/cpl/record/index.xsql?pubid=93674>).

Home page: <http://www.cs.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.

Student Projects Masters students Anders Karlsson and Tobias Olausson are currently developing a (fast) mp3 decoder in Haskell. More info about this project can be found at <http://trac.haskell.org/HQmpd>. The

project will also include a showcase player using the decoder.