Report on IFL 2002

Olaf Chitil

September 24, 2002

IFL 2002 took place 16–18 September 2002 in Madrid. Here are some personal comments on some talks.

I do not see any major new ideas or directions. Mostly people continue what they did before. Here I use my own classification scheme.

Contents

1 Parallel and Distributed Implementations

Porting the Eden System to GHC 5.00 [Jost Berthold, Rita Loogen, Steffen Priebe, and Nils Weskamp]

Very technical. Only proves that implementing a language by modifying an existing language implementation such as ghc is a bad idea. Trying to understand a complex implementation and keeping up with development takes most of the time.

Thread Migration in a Parallel Graph Reducer [Andre Rauber Du Bois, Hans-Wolfgang Loidl, and Phil Trinder]

Technical and low-level.

Paging Lists in a Distributed Virtual Memory system for a Functional Language Using Heap-Exclusive Frames [Marco Morazan and Douglas Troeger]

Technical and low-level.

Towards a Strongly Typed Functional Operating System [Arjen van Weelden and Rinus Plasmeijer]

The title is a bit misleading. A library for distributed programming in Clean, especially process scheduling and type-safe communication is presented. Quite technical.

A Prototype of CPPCC – Safe Functional Mobile Code in Clean [Karoly Daxkobler, Zoltan Horvath and Tamas Kozsik]

An architecture for extending mobile code with properties that the receiver can certify using the clean prover Sparkle. Leaves lots of open questions in my mind.

Global Scheduler Properties arrived from Local Functions [Juan Penas]

(No paper) In a project with Ericsson the dynamic process development of Erlang programs was analysed.

2 Type-Based Analysis'

Cost Analysis Using Automatic Size and Time Inference [Alvaro Rebon Portillo, Kevin Hammond, and Hans-Wolfgang Loid]

Aim is granularity analysis for implicit parallelism in a strict functional language. Yet another extension of the sized types of Hughes et al. No proofs at all; no reference to an operational semantics to prove correct approximation of costs nor correctness of presented type inference algorithm. Open how good the approximations are in practice (no implementation).

An algorithm for inferring array types with varying levels of shape information [Sven-Bodo Scholz]

The type system was already presented last year and in the talk he did not say that much about the new inference algorithm.

3 Compiler Optimisations

Optimistic Evaluation [Robert Ennals and Simon Peyton Jones]

Haskell is evaluated eagerly. Long eager computations are aborted. Runtime profiling keeps track where eager evaluation is worthwhile and where not. The implementation seems to have required major modifications/extensions of ghc. A few programs get slower and in principle unexpected non-termination is still possible. However, general speed-up is 5-25% and hence it will come into the standard ghc distribution.

Selective Tail Call Elimination [Yasuhiko Minamide]

Some implementation platforms such as the Java Virtual Machine do not directly support tail calls. Then these can be implemented by using a 'trampoline'. However, the trampoline costs runtime and here a type analysis is presented to determine where trampolines are needed to assure limited stack use by tail calls. Not complicated, but seems to be well presented. (the language is strict)

Fusion in Practice [Diederik van Arkel, John van Groningen, and Sjaak Smetsers]
Deforestation has been implemented in Clean. The algorithm is an improved version of Chin's (based on Wadler's) fold-unfold algorithm. Mainly unfolding of consumers is more restricted to unfold only when this is likely to permit a later folding step. Accumulating parameters are often a problem, but on the other hand the algorithm also covers specialisation of higher-order functions, overloaded functions and generic functions. Not many measurements yet.

Compilation of Floating Point Arithmetic in the Uncooperative Erlang Environment [Tobias Lindahl and Konstantinos Sagonas]
A straightforward simple type-based analysis and technical details.

Fast Functional Lists [Phil Bagwell]
An implementation of a list as a sequence of arrays of different sizes. Most O(n) functions become O(log(n)). Not considered as a data structure implemented in a functional language, but at a very low level. A toy Lisp interpreter using this kind of lists was implemented. List benchmarks perform better with this interpreter than in OCaml, but is this a meaningful comparison?

4 Programming Tools

Parametric Type Inferencing for Helium [Bastiaan Heeren and Jurriaan Hage]
Helium is a compiler for a subset of Haskell. It is used in teaching to avoid the complexities of full Haskell. The type inference algorithm first collects all type variable constraints and subsequently can solve them in various different orders. Thus different type error messages can be produced for one program and this year the authors intend to experiment with their students to see which error messages are best. Lacks comparison with some closely related work (e.g. by Bruce McAdams at Edinburgh).

Transforming Haskell for Tracing [Olaf Chitil, Colin Runciman, and Malcolm Wallace]
Obviously the highlight of the workshop ;-)

Deriving Animations from Recursive Definitions [Alcino Cunha, Jose Barros, and Joao Saraiva]
Aim: debugging and understanding functional programs. The animation grows the call tree of a recursive definition, the nodes are later replaced by the call results. The basis is a program transformation that makes the call tree explicit as a data structure. In some way reverse deforestation is at the core. The transformation makes use of the theory of hylomorphisms, their monadic versions and generic Haskell. Fascinating how all these complex theoretical works are combined in a new context, but the animations are quite poor and surely could be obtained much more easily. We could provide an improved variant for such animations as a viewing tool for Hat.

A Debugger for Concurrent Haskell [Frank Huch and Thomas Böttcher]

Step-by-step debugging of concurrent threads and their communciation. Scheduling can be influenced. Because all operations are in the IO monad, lazyiness is not an issue. Simply replaces the original concurrency library by a wrapped variant. Limited functionality, but simple implementation in return.

A generic test-system [Pieter Koopman]

(no paper) Basically QuickCheck for Clean. Uses Clean's generic definitions to define test data generation once for all types. Also test data generation less based on random generation but more on systematic enumeration of values.

5 Specific Application Domains

Introducing generalized rank conjunctions to SAC or how to write and compile nested array operations [Clemens Grelck and Sven-Bodo Scholz]

A new language feature for arrays looking a bit like list comprehensions. Both design and efficient implementation are discussed.

Concurrency abstractions for Concurrent Haskell [Volker Stolz and Frank Huch]

A library for higher-level concurrency constructs, mostly based on Erlang.

FunWorlds/HOpenGL – Functional Programming and Virtual Worlds [Claus Reinke]

A variant of functional reactive animation (FRAN) based on HOpenGL. Slightly different combinators with slightly restricted expressiveness are used which promise a simpler implementation with predictable performance. A prototype shall be released soon.

Predictable Space Behaviour in FSM-Hume [Kevin Hammond and Greg Michaelson]

Hume is intended to be a high-level strict functional language for hard real-time programming. FSM-Hume is a sublanguage without function definitions based on finite state automata. A sized type system is used to predict the space behaviour. The talk did not explain the language concepts very well. Kevin is quite interested in solving the real-time tasks of our real-time system course (marble sorter, elevator) in Hume.

6 Program Development and Verification

Proving Make Correct: IO Proofs in Haskell and Clean [Andrew Butterfield, Malcolm Dowse and Glenn Strong]

Continuation of last year's paper. I don't like modelling IO as passing the world around. The simplistic model makes lots of assumptions that should be spelt out explicitly.

Parallel Programming via Stepwise Refinement with Abstract Parallel Machines [Nils Ellmenreich and Christian Lengauer]

A kind of software engineering method for stepwise refinement for parallel programming. I didn't understand much, but it is not my area and it was the last talk of the day.

From requirements to functional programs — the HasCasl development paradigm [Christian Maeder, Till Mossakowski, and Lutz Schröder]

HasCasl is an algebraic specification language that is influenced by Haskell, in particular it has type classes. Hardly about functional programming, except that the example was the derivation of the SECD machine, impossible to follow in a talk.

Reasoning about explicit strictness in a lazy language using mixed lazy/strict semantics [Marko van Eekelen and Maarten de Mol]

The central theme is the value bottom, but several aspects seem to be mixed. The aim is to allow the prover Sparkle to reason about properties taking bottom into account. The authors note that the existence of polymorphic explicit strictness annotations change the semantics of a lazy language. Hence they extend Launchbury's semantics with a strict let. The relationships between several semantics are not really surprising.

7 Genericity, Dynamic Types

Lazy Dynamic Input/Output in the lazy functional language Clean [Martijn Vervoort and Rinus Plasmeijer]

The implementation for reading and writing values of dynamic type. Because these values can contain closures, dynamic linking to compiled Clean code is needed. Technical.

Dynamic Values Need Polytypic Functions [Peter Achten, Artem Alimarine and Rinus Plasmeijer]

Discusses the interaction between dynamic types and genericity in Clean. In practice they do not work together as they probably should. Some pragmatic tricks are presented to partially solve the problems.

Strategic polymorphism requires just two combinators! [Ralf Laemmel and Joost Visser]

Implementation of some form of genericity that the listener was supposed to know about. Rather technical. May be implemented in ghc, because the implementation is supposedly simple.

8 The Rest

A program transformation for implementing on-demand strategy annotations [Maria Alpuente, Santiago Escobar and Salvador Lucas]

The algebraic language OBJ allows annotating function definitions with local strategies. Basically they state when which function argument shall be evaluated. The authors presented an extended annotation scheme in another paper and here they show how a program with extended annotations can be transformed into a semantically equivalent program with the existing annotation scheme.

The Unified Model of Computation as an Implementation Platform for Functional Languages [Alex Ferguson and John Morrison]

This model shall unify data-driven, demand-driven and imperative computation in an inherently parallel formalism based on graphs. Was sold as the new paradigm that everyone should know about and it was said that a dozen people in Cork work on it. Fancy diagrams and new terminology, but I didn't understand this unified model (which doesn't yet have a semantics) at all.

An Abstract Lambda-Calculus Machine [Werner Kluge]
The talk did not take place because W Kluge was ill. The paper presents a machine for rewriting to normal form, not just head normal form, and seems to be rather historical.

Haxcel: A Spreadsheet Interface to Haskell [Björn Lisper and Johan Malmström]
This is not the first spreadsheet with a lazy functional language as formula language. The advantages to previous approaches are unclear. The implementation is compiler independent, similar to hi, but it is only a limited prototype.

Development of a generic e-learning Multiparadigm Programming Language System [Jose Labra, Jose Morales, Roberto Turrado, Alfredo Jarero]
A web based system for teaching programming in various languages (Haskell, Prolog,...). Naturally based on XML. Very basic functionality (administration of students, web interface to interpreter), but will be used in practice this year.

Functional Transformation of Circular Data [Baltasar Trancon y Widemann]
A stack machine for strict evaluation of a certain kind of functions on coalgebraically defined circular data. Lots of category theory and bullets killed the audience.

9 Further Remarks

Michael Hanus and Frank Huch at Kiel work together with Maria Alpuente and German Vidal at Valencia on a debugger for Curry. Up till now they formulated an operational semantics with sharing for Curry (based on Launchbury's).

Axel Simon (a former student of mine) is doing a PhD at the University of Kent under the supervison of Andy King. They are using abstract interpretation to find (or prove the absence) of errors in C programs, especially those related to pointers. Within one year they already published two papers.