Invited Talk

Why Dependent Types Matter

James McKinna
University of St. Andrews
Scotland
james.mckinna@st-andrews.ac.uk

Abstract
Language designers have in recent years proposed a wealth of richer type systems for programming which seek to extend the range of statically enforced guarantees on data and code. Most such proposals have been evolutionary extensions of ML or Haskell, offering programmers a balanced compromise between expressive strength and existing well-understood technology. Typically they revolve around type- or kind-indexed types such as GADTs, supported by limited equality reasoning at the type-checking level, thus separating the dynamic behaviour of programs from the (simpler) static behaviour of indexing information occurring in their types.

I want to argue in this talk for a more radical departure from such practice by examining full spectrum type dependency, lifting such restrictions on the data upon which types may depend. Conor McBride and I designed the language EPIGRAM for experiments in programming with inductive families of data (of which GADTs are a special case). Using it for illustration, I will explore some of the possibilities and challenges afforded by full spectrum type dependency at the static and dynamic level:

• types directly support modelling complex invariants in terms of other data (rather than their types), with a Curry-Howard flavour of data-as-evidence; such complexity is on a 'pay-as-you-go' basis, while keeping type annotations and other syntactic overheads to a minimum;
• data decomposition steps, e.g. case analysis, furnish more informative interactions between types and values during typechecking; such steps may moreover be abstractly specified by their types, and thus user definable; this supports a style of programming embracing 'learning by testing', views, and Burstall's 'hand simulation plus a little induction';
• the absence of a rigid phase distinction need not lead to type-passing or excessive run-time overhead; effectful computation, in particular partiality, can be incorporated via variations on existing ideas such as monads.

This talk is based on joint work with Conor McBride, Edwin Brady and Thorsten Altenkirch.

Categories & Subject Descriptors: D.1.1, D.3.2 Functional programming, D.3.3 Data types and structures, F.3.1 Specifying, Verifying and Reasoning about Programs, F.3.3 Type structure.


Bio
The speaker is a Senior Teaching Fellow in the School of Computer Science at the University of St. Andrews. He took his Ph.D. in Computer Science in the LFCS, Edinburgh under Rod Burstall. He was previously a lecturer at the University of Durham and a researcher at LFCS. His primary technical interest is the design and use of advanced type systems for programming and proof. His previous work includes an early approach to certified code, deliverables, with Burstall, and machine-checking the meta-theory of typed lambda calculi with name-carrying syntax, with Randy Pollack. His website is http://www.dcs.st-and.ac.uk/~james/.