Typed Boa Calculus

Barry Jay[1]*

Centre for Quantum Computing and Intelligent Systems, School of Software,
University of Technology, Sydney Barry.Jay@uts.edu.au

Queries such as select are the mainstay of database programming languages such as SQL where they are typically applied to tables of records. Generic queries apply to a wider class of data structures, such as pairs, list and trees. Various mechanisms have been developed to support such generality, such as Scrap-Your-Boilerplate, Stratego, pattern calculus and factorisation calculus. The latter two are relatively general and powerful because they employ a strongly-typed, confluent rewriting system. The boa-calculus (Joint with Jose Vergara) builds on the factorization calculus to support generic queries which can act on lambda abstractions. That is, it supports both the beta-reduction of lambda calculus and the reduction of combinators.

The main challenge is to type the intensional operators F (for factorisation) and E (for equality). The type of F has been presented elsewhere, namely

$$F : \forall X.\forall Y.X \to Y \to (\forall Z.(Z \to X) \to Z \to Y) \to Y \ .$$

The type of E improves on earlier work with Jens Palsberg to be

$$E : \forall X^\circ.\forall Y.\forall Z.X^\circ \to Y \to [X^\circ \prec Y]Z \to Z \to Z$$

where X° is an *operator type variable* and $[X^\circ \prec Y]Z$ is a *constrained type*. The operator type variables represent *operator types*. The use of X° above ensures that if it becomes the type of some operator O then it becomes $\mathsf{Ty}[O]$. Thus, instantiating X° and Y and Z yields a type of the form

$$E : \mathsf{Ty}[O] \to U \to [\mathsf{Ty}[O] \prec U]T \to T \to T \ .$$

The main technical challenge is to control the use of the type constraints $\mathsf{Ty}[O] \prec U$. The talk will present the latest version of this developing type system.