Typed Tagless Final Interpreters

Oleg Kiselyov

oleg@okmij.org

Abstract. The so-called 'typed tagless final' approach of Carette et al. [6] has collected and polished a number of techniques for representing typed higher-order languages in a typed metalanguage, along with type-preserving interpretation, compilation and partial evaluation. The approach is an alternative to the traditional, or 'initial' encoding of an object language as a (generalized) algebraic data type. Both approaches permit multiple interpretations of an expression, to evaluate it, pretty-print, etc. The final encoding represents all and only typed object terms without resorting to generalized algebraic data types, dependent or other fancy types. The final encoding lets us add new language forms and interpretations without breaking the existing terms and interpreters.
These lecture notes introduce the final approach slowly and in detail, highlighting extensibility, the solution to the expression problem, and the seemingly impossible pattern-matching. We develop the approach further, to type-safe cast, run-time-type representation, Dynamics, and type reconstruction. We finish with telling examples of type-directed partial evaluation and encodings of type-and-effect systems and linear lambda-calculus.

1 Introduction

One reinvents generic programming when writing accumulation, pretty-printing, equality comparison functions for data types – and writing these functions again and again for extended or slightly different data types. Generic programming aims to relieve the tedium by making programs more applicable, abstracting over values, shapes, processing strategies and so on. One may dually view a data type as an encoding of a domain-specific language, and data type processing as an interpretation of that language. That view comes to the fore if the data type indeed represents an abstract syntax tree (AST). Generic programming then is writing extensible interpreters. The embedded-language point-of-view highlights that oftentimes not all sentences generated by a context-free grammar – not all values fitting the datatype declaration – are regarded as meaningful. A type system is a common way of stating the additional validity constraints. Typed extensible interpreters of typed languages, fitting the theme of the school, express both generic programming (parametrization over interpretations) and indexed programming (expressing processing invariants and validity constraints).

There are two basic approaches to embedding languages and writing their interpreters, which we shall call, somewhat informally, 'initial' and 'final'. The initial approach represents a term of an object language as a value of an algebraic