Tabular Expressions and Total Functional Programming

Baltasar Trancón y Widemann
David Lorge Parnas

Software Quality Research Laboratory (SQRL)
University of Limerick, Ireland

Our research Group is developing methods of producing practical reference documentation for software products and components. Our document contents are defined by a relational model in which each document is required to be a representation of a specified relation. In effect, we are using mathematical descriptions of relations to provide specifications and descriptions of programs written in conventional languages.

Key to making these documents readable is a multidimensional form of expressions, which we call tabular expressions. These parse complex expressions into arrays of simpler expressions allowing readers to "look up" the information that they seek without understanding the whole expression.

Tools that check and evaluate these expressions would be very useful when these methods are applied and we are looking for efficient implementations of such functions.

Typed lambda calculus has been used as a foundational model of general mathematical expressions in early mechanized mathematics systems such as *Automath*, and in theorem provers such as *Coq*, *HOL* and *Isabelle*. We explore the application of this approach to interpret tabular expressions. Our approach is based on a simple functional definition language for tables, their semantics and auxiliary functions and datatypes. This language can be compiled to produce evaluators, or interpreted to interoperate with a theorem prover or computer algebra system for symbolic reasoning. We have implemented a front-end for the language, a type checker and a back-end that generates *Java* code.

In mathematics, total functions are often preferred over partial ones, because algebraical and logical reasoning seems easier for the former than for the latter. The denotational semantics of *Turing*-complete functional programming languages, on the other hand, treat all functions as potentially partial. This is reflected in the type system, and valuable totality information is lost in the implementation of a function in such a language.

We have chosen a total functional system based on *Barendregt's lambda cube* for both reasoning and execution purposes. The experience we have gained from the implementation of basic standard functions and data structures and several types of tables, suggests that the restrictions on recursion and type inference imposed by such a system do not impede practical programming unduly. We show that many aspects of the modeled theory map directly to standard design principles of functional programming such as *folds* and *monads*, and result in highly concise, explicit and reusable definitions.

References

[1] D. L. Parnas, J. Madey, and M. Iglewski. Precise documentation of well-structured programs. *IEEE Trans. Softw. Eng.*, 20(12):948–976, 1994.