JFP **27**, e15, 2 pages, 2017.
doi:10.1017/S0956796817000065

Special issue on Programming with Dependent Types Editorial

There has been sustained interest in functional programming languages with dependent types in recent years. The foundations of dependently typed programming can be traced back to Martin–Löf's work in the 1970s. In the past decades, this vision has given rise to the development of proof assistants and functional programming languages based on dependent types. The increased popularity of systems such as Agda, Coq, Idris, and many others, reflects the growing momentum in this research area. After sending out our first call for papers in October 2015, we are happy to accept six articles in this special issue covering a wide spectrum of topics.

Despite their theoretical appeal, there are very few examples of programming languages based on dependent types which have been used to construct applications with a graphical user interface. Abel *et al.* show how to write such GUIs in Agda in their article *Interactive Programming in Agda – Objects and Graphical User Interfaces*.

Programming languages with dependent types provide a rich design space for describing data types that capture invariants precisely. One drawback, however, is that there may be many subtle variations of the same data type, resulting in duplicated functions for each such variation. *Ornaments* provide a language for describing the relationship between data types. Ko and Gibbons's article *Programming with Ornaments* shows illustrative examples of ornaments in action; Dagand's article *The Essence of Ornaments* gives a novel description of ornaments in terms of many-sorted signatures.

A higher order unification algorithm lies at the heart of many implementations of dependently typed programming languages. Ziliani and Sozeau present a new such algorithm for the Calculus of Inductive Constructions in their article *A Comprehensible Guide to a New Unifier for CIC Including Universe Polymorphism and Overloading*. This algorithm both provides useful heuristics and deals with several of the features of the calculus specific to the Coq system.

Stump addresses another central concept of dependently typed programming languages in his article *The Calculus of Dependent Lambda Eliminations*: the notion of inductively defined data type. One of the motivations for the original Calculus of Constructions was the use of impredicative quantification for compactly and elegantly encoding data types. Such encodings, however, were soon discovered to have several drawbacks. To remedy these the calculus was extended with a system of primitive inductive data types in the Calculus of Inductive Constructions. With his Calculus of Dependent Lambda Eliminations, Stump proposes another alternative: a new strengthened system based on impredicative encodings, but with

certain constructor constraints, aimed at eliminating the shortcomings of the original impredicative encodings.

As mentioned above, dependently typed programming languages evolved from Martin–Löf's foundational theories for constructive mathematics. Perhaps surprisingly, the definition of finiteness in such theories is quite subtle. For instance, there are several definitions of the notion of finite set that are not equivalent. Uustalu and Veltri's article *Finiteness and Rational Sequences, Constructively* studies the definition of the related notion of rational (or ultimately periodic) sequences and presents several different implementations of this notion in Martin–Löf type theory.

We would like to thank the authors for submitting their work to JFP and their patience with the reviewing process, the reviewers for their diligence, and finally, the JFP editors, Jeremy Gibbons and Matthias Felleisen, who were incredibly supportive and encouraging throughout the editorial process.

Wouter Swierstra
Department of Information and
Computing Science University of Utrecht
w.s.swierstra@uu.nl

Peter Dybjer
Department of Computing Science and
Engineering Chalmers University of Technology
peterd@chalmers.se