Constructive Mathematics and Functional Programming (Abstract)

Thierry Coquand

Department of Computer Science and Engineering,
Göteborg University and Chalmers University of Technology
Sweden
coquand@cs.chalmers.se

Around thirty years ago, P. Martin-Löf [12] suggested that the intuitionistic theory of types, originally designed as a formal system for constructive mathematics, could be viewed as a programming language. The conclusion of this paper stresses the mutual benefit of relating constructive mathematics and computer programming. In one direction one gets a precise system of notations for both statements and proofs, and one obtains the computerization of abstract intuitionistic mathematics that was asked by Bishop [2]. In the other direction, computer programming "gets access to the whole conceptual apparatus of pure mathematics".

In the first part of this talk we shall survey some recent works that illustrate this relation and its fruitfulness. One line of work, close to Bishop, represents real numbers and numerical functions [4,13] in type theory. Another line is concerned with algorithms on finite combinatorial structure (graphs, hypermaps, finite groups). One main example is the complete formalization of a proof of the four color theorem by G. Gonthier and B. Werner. The report on this work [9] points out as well the mutual benefits of this correspondence: "Although this work is purportedly about using computer programming to help doing mathematics, we expect that most of its fallout will be in the reverse direction - using mathematics to help programming computers". A related work, also dealing with hypermaps, aims at obtaining formal specification in geometric modeling [6], and presents algorithms that can be designed in this way [7]. There is also on-going work [8] on the formalization of finite group theory.

The second part will reflect on the design of type theory as a *functional* programming language. The analogy between type theory and functional programming was pointed out already by Martin-Löf [12] (for instance the correspondence between canonical and non-canonical form of expressions and the notion of constructors and selectors, respectively, of Landin [11]). This analogy should go further and type theory should benefit in using more the powerful system of notations provided by functional programming. (In particular, *where expressions* correspond to local abbreviations, definitions and lemmas, functions defined by *pattern-matching* correspond to definitions by case and proofs by case analysis, uses of *recursive definitions* correspond to inductive arguments, *module systems* can be used to structure proofs; the system Agda [1] follows these analogies.)

S. Drossopoulou (Ed.): ESOP 2008, LNCS 4960, pp. 146–147, 2008.

The work of B. Gregoire and X. Leroy [10] illustrates well also this analogy by showing how an evaluation machine for functional programming can be modified in a simple way to provide an efficient algorithm for testing convertibility in type theory. We explain finally how we can precise further the representation of type theory as a functional programming language using some recent results in domain theory [3,5].

References

1. Agda home page, `http://appserv.cs.chalmers.se/users/ulfn/wiki/agda.php`
2. Bishop, E.: Mathematics as a numerical language. In: 1970 Intuitionism and Proof Theory (Proc. Conf., Buffalo, N.Y., 1968), pp. 53–71. North-Holland, Amsterdam (1970)
3. Berger, U.: Continuous Semantics for Strong Normalization. CiE, 23-34 (2005)
4. Bertot, Y.: Affine functions and series with co-inductive real numbers. Mathematical Structures in Computer Science 17(1), 37–63 (2007)
5. Coquand, T., Spiwack, A.: A proof of strong normalisation using domain theory. LMCS 3(4:12) (2007)
6. Dufourd, J.-F.: A hypermap framework for computer-aided proofs in surface subdivisions: genus theorem and Euler's formula. In: SAC 2007, pp. 757–761 (2007)
7. Dufourd, J.-F.: Design and formal proof of a new optimal image segmentation program with hypermaps. Pattern Recognition 40(11), 2974–2993 (2007)
8. Gonthier, G., Mahboubi, A., Rideau, L., Tassi, E., Théry, L.: A Modular Formalisation of Finite Group Theory. In: Schneider, K., Brandt, J. (eds.) TPHOLs 2007. LNCS, vol. 4732, pp. 86–101. Springer, Heidelberg (2007)
9. Gonthier, G.: A computer-checked proof of the Four Colour Theorem. (unpublished)
10. Grégoire, B., Leroy, X.: A compiled implementation of strong reduction. In: ICFP 2002, pp. 235–246 (2002)
11. Landin, P.J.: The mechanical evaluation of expressions. Computer Journal 6(4) (1964)
12. Martin-Löf, P.: Constructive mathematics and computer programming. In: Logic, methodology and philosophy of science, VI (Hannover, 1979), Stud. Logic Found. Math., 104, pp. 153–175. North-Holland, Amsterdam (1982)
13. O'Connor, R.: A monadic, functional implementation of real numbers. Mathematical Structures in Computer Science 17(1), 129–159 (2007)