

On dependent types and intuitionism in programming mathematics

Sergei D. Meshveliani *

Program Systems Institute of Russian Academy of sciences,
Pereslavl-Zalessky, Russia. email: mechvel@botik.ru

Abstract. It is discussed a practical possibility of a provable programming of mathematics basing on intuitionism and the dependent types feature of a programming language. The principles of constructive mathematics and provable programming are illustrated with examples taken from algebra. The discourse follows the experience in designing in **Agda** a computer algebra library **DoCon-A**, which deals with generic algebraic structures and also provides the needed machine-checked proofs.

This paper is a revised translation of a certain paper published in Russian in 2014.

Keywords: constructive mathematics, computer algebra, machine-checked proof, dependent types, Agda

1 Introduction

This paper is a certain revised translation of the paper [12] published in Russian in 2014.

It describes the experience in searching for the most appropriate tool for programming mathematical computation. The investigation is based on the practice of programming symbolic computations in algebra.

The goal was to find an universal programming language with possibly rich mathematical expressiveness, to explain its advantages with respect to other languages, and to test this tool on implementing a considerable piece of a real-world computer algebra.

The history of this search and program design experience consists of the three main steps. They correspond to the three language classes:

1. generic programming languages,
2. purely functional languages,
3. languages with dependent types.

Below it is explained of why the feature (1) is important. For example, the **Haskell** language [8] belongs to the classes (1) and (2). In this language the author has implemented in 1990-es the **DoCon** library for commutative algebra [13] [14]. It is presumed here that the notion of pure functionality, and the main features of the **Haskell** language, are known to the reader.

Finally, we consider the **Agda** language [1] [18], which belongs to the classes (1), (2), (3). Developing a workable implementation for such a complex language is a great technical problem. Currently, **Agda** is, mainly, a workable tool (with a great field remaining for desirable optimizations). In this paper we try to show that **Agda** fits best the needs of programming computation in mathematics.

About the sources on mathematics: the mathematical notions used in this paper can be found in [20] (algebra) and in [7], [5] (computer algebra).

The following discourse concerns mainly the ways to program mathematical computations and proofs in the **Agda** language.

* This work is supported by the FANO project of the Russian Academy of Sciences, the project registration No AAAA-A16-116021760039-0.