Maciej Bendkowski

Theoretical Computer Science Department,
Faculty of Mathematics and Computer Science,
Jagiellonian University, ul. Łojasiewicza 6, 30–348 Kraków

Quantitative aspects of computational complexity in lambda calculus

Description for the general public

With the dawn of modern computer science in the 1930s, multiple mathematical theories emerged aiming at formalizing the notion of *computation*, such as the famous Turing machines, μ-recursive functions or lambda calculus. Although all were proven equivalent, each one of them proposed a different approach to the nature of computability. One prominent example is lambda calculus, developed by the American mathematician and computer science pioneer – Alonzo Church. Though originally meant as an alternative foundation of mathematics, lambda calculus quickly became popular in the emerging computer science, expressing the theoretical capabilities of modern computers. Nowadays, lambda calculus serves not only as the theoretical foundations of computer science, but is also used in practical applications including artificial intelligence, automated theorem proving, formal software verification and many more.

Despite its age, lambda calculus is still actively studied. Looking at programs expressed in the language of lambda calculus as combinatorial objects, we ask the natural question about their typical behavior and properties. Suppose we have in mind a fixed property P. In our line of research, we ask about the limit behavior of the fraction of programs of size n satisfying P in the set of programs of size n. If this limit exists, we have the asymptotic density of programs satisfying P – the main tool in studying properties of typical programs of lambda calculus. Of course, the most interesting properties we ask about are semantic in nature, i.e. properties which do not explicitly depend on the program syntax. A famous example of this type of properties is the *halting property* asserting that the represented computations will eventually stop. Previous research uncovered a striking discrepancy of this property depending on the program representation. And so, based on the assumed model, asymptotically almost all programs halt, or there exist a non-trivial fraction of programs looping ad infinitum.

It is worth noticing that quantitative investigations are not entirely theoretical, but have interesting practical applications, including software verification. In recent years, we observe an increasing popularity of software testing involving random computer-generated data. Program subclasses satisfying requested asymptotic properties yield new tools in this technique of software testing used, e.g. in testing functional programming language compilers.

In our project we propose to study the computational complexity distribution of typical lambda terms. We plan on investigating the impact of substitution on the average time complexity of represented programs. We wish to compare different evaluation strategies in the case of their typical behavior. As proven by our previous research, this approach allows to find new connections between semantic properties of programs and their syntactic representation, raising hopes for a deeper understanding of the typical computational complexity of random lambda terms.