Research on Per Martin-Löf's type theory

G. Marikyan
State University of New York Empire State College, USA
e-mail: `Gohar.Marikyan@esc.edu`

The purpose of this article is to discuss my research of Martin-Löf's type theory. It consists of a number of scholastic proofs and results. In order to use a theory it is a requirement to prove its consistency. One of my results is a proof of consistency of Martin-Löf's type theory. It has been submitted for publication in "Springer Archive for Mathematical Logic". Another problem is the efficiency of the generated computer programs. This problem can be reformulated into a problem of comparison of complexity of inferences in the type theory and in other known systems, in the Hilbert-type System, and in the Gentzen-type System.

I will present an overview of my algorithm that automates the construction of an object of the type that represents the problem under question. The construction of this algorithm raises fixed-point type problems. I presented my algorithm in 1986 at the "Personal Computers & Local Networks" conference in Georgia.

Keywords: type theory, automation of inherence.

1. Consistency of Martin-Löf's type theory

The most important property of any theory is its consistency. If the theory is not consistent, then there exist a proposition A that both A and $\neg A$ are provable in the theory. Therefore, a not consistent theory cannot be used for automated computer program writing. Simply put, a not consistent theory is useless.

I have proved the consistency of Martin-Löf's type theory. I have presented my proof at 2005–2006 Association for Symbolic Logic Winter Meeting (with Joint Mathematics Meetings) San Antonio, Texas, January 14–15, 2006, and has been submitted to "Springer Archive for Mathematical Logic" for publication.

2. Complexity of logical inferences in Martin-Löf's type theory

To characterize the complexity of inferences in Martin-Löf's type theory [1] we need to compare complexity of inferences in the type theory with complexity of inferences in other, already researched, formal systems. In order to compare complexity of inferences in different systems we will need to define methods of complexity measurements universal for all of these systems. I have compared complexity of inferences in Martin-Löf's small types [2], that is, types of V_0, with complexity of inferences in two other well known systems. One of these systems is the Hilbert-type Intuitionistic Formal System (hereafter H) [3]. The second system is a Gentzen-type Intuitionistic Formal System (hereafter G) [3, 4]. All these three systems (small types, H and G) are formalizations of arithmetic. Also, I have defined three measurements of complexity, and Shannon's Functions [5] for all these measurements of complexity. Then I have defined upper and lower limits of these Shannon's Functions. My results show that complexity of inferences in small types are not more complex than complexity of inferences in both H and G. (I am in the process of preparing these results plus additional results on complexity of inferences for publication.) My overall conclusion is that inferences in Martin-Löf's type theory are definitely less complex than inferences in both H and G.

3. Automation of inference in Martin-Löf's type theory

I have built an algorithm "Inference Research System with Dialogue in Martin-Löf's Intuitionistic Type Theory" that automates the construction of an object of the type that represents the problem under question [6]. The construction of this algorithm raises fixed-point type problems. I presented my algorithm in 1986 at the "Personal Computers & Local Networks" conference in Georgia. Later, on the basis of Martin-Löf's type theory a few systems such as NuPRL, Lego, Coq, Agda, ALF, Twelf and Epigram, were constructed. Here are brief descriptions of these systems taken mainly from Wikipedia.

Agda is an interactive system for developing constructive proofs in a variant of Per Martin-Löf's type theory. In Agda the proof is a term, not a script. It can also be seen as a functional programming language with dependent types.

Coq is a proof assistant application. It is not an automated theorem prover but includes automatic theorem proving tactics and various decision procedures.

Epigram is a functional programming language with dependent types. The goal is to support a smooth transition from ordinary programming to integrated programs and proofs whose correctness can be checked and certified by the compiler. Epigram exploits the propositions as types principle, and is based on intuitionistic type theory.

Lego Mindstorms is a line of Lego sets combining programmable bricks with electric motors, sensors, Lego bricks, and Lego Technic pieces. The first retail version of Lego Mindstorms was released in 1998. The current version is Lego Mindstorms NXT which is a programmable robotics kit released by Lego in late July 2006.

NuPRL is a higher-order proof development system developed at Cornell University.

Twelf is an implementation of the logical framework. It is used for logic programming and for the formalization of programming language theory.

Let us use an example to illustrate how the language of Martin-Löf's type theory works. As an example we will use the following formula:

$$\left(\sum t \in C(0/n)\right)\left(\left(\prod n \in N\right)\left(\prod y \in C\right) C(n'/n)\right) \rightarrow \left(\left(\prod n \in N\right) C\right) \tag{1}$$

where N is type of natural numbers, and the variables n and y occur free in C (1) is a type. I will show how to construct an object of this type. This object may be interpreted as a proof of (1). Assume

$$n \in N, \tag{2}$$

$$y \in C, \tag{3}$$

$$z \in \left(\sum t \in C(0/n)\right)\left(\left(\prod n \in N\right)\left(\prod y \in C\right) C(n'/n)\right) \tag{4}$$

where (2) means that n is a variable of type N, and (3) means that y is a variable of type C. To prove (1) we need to construct an object of the type to the right of the arrow ($\rightarrow$) using the assumption (4).

First we have to apply the $\sum$-elimination rule to (4). Let us use the following abbreviations:

$$p(c) \text{ for } (Ea, b)(c, a), \quad q(c) \text{ for } (Ea, b)(c, b)$$

which means that $c = (p(c), q(c))$. By $\sum$-elimination we get

$$p(z) \in C(0/n) \tag{5}$$

and

$$q(z) \in \left(\prod n \in N\right) \left(\prod y \in C\right) C(n'/n). \tag{6}$$

By $\prod$-elimination, from (6) and (2) we get

$$q(z) = q(z)(n/n) \in \left(\prod y \in C\right) C(n'/n). \tag{7}$$

By $\prod$-elimination, from (7) and (3) we get

$$q(z)(y) \in C(n'/n). \tag{8}$$

By N-elimination, from (2), (5) and (8) we get

$$(Ru, v)(n, p(z), q(z)(y)) \in C(n/n) = C. \tag{9}$$

Finally, by $\prod$-introduction, from (9) and (4) we get

$$(\lambda z)(Ru, v)(n, p(z), q(z)(y)) \in$$

$$\in \left[\left(\sum t \in C(0/n)\right) \left(\left(\prod n \in N\right) \left(\prod y \in C\right) C(n'/n)\right) \rightarrow \left(\left(\prod n \in N\right) C\right)\right].$$

Thus,

$$(\lambda z)(Ru, v)(n, p(z), q(z)(y))$$

is the sought object of the type (1).

This procedure of constructing an object offers an idea about how to choose which rule to apply. The ultimate operation of (1)

$$\left(\sum t \in C(0/n)\right) \left(\left(\prod n \in N\right) \left(\prod y \in C\right) C(n'/n)\right) \rightarrow \left(\left(\prod n \in N\right) C\right)$$

is $\rightarrow$, that is, $\prod$. Notice that the assumption (4) was made in order to construct an object of (1) by using $\prod$-introduction. Then we used elimination rules to go from outside to inside to construct objects of types. Next, to construct an object of the type to the right from the arrow, we applied introduction rules from inside, out. Finally, we applied the $\prod$-introduction rule to construct an object of the type (1). This logic can be used to any type, and construct an object of that type, if there is one. This method can be used to build an algorithm that automates inference in Martin-Löf's type theory. The algorithm I have built [6] is a system with dialog. When applied to a type it constructs an object of that type. This system can be used as an algorithm that solves the proposition. Based on this algorithm a computer program can be created. My system with dialog can be used as a skeleton of a software that automatically synthesizes computer programs.

References

[1] Per Martin-Löf. Constructive mathematics and computer programming // Sixth Intern. Congress for Logic, Methodology, and Philosophy of Science. Amsterdam: North-Holland, 1982. P. 153–175.

[2] Per Martin-Löf. An intuitionistic theory of types: predicative part // Logic Colloquium '73, H.E. Rose, J.C. Shepherdson. Amsterdam: North-Holland, 1973. P 73–118.

[3] Kleene S.C. Mathematical Logic. N.Y., L., Sydney: John Wiley and Sons, 1967.

[4] Kleene S.C. Introduction to Metamathematics. Vol. 1. Wolters-Noordhoff Publishing and North-Holland Publishing Company, 1971.

[5] Goodstein R.L. Mathematical Logic. Leicester: University Press, 1957.

[6] Marikyan G. Inference research system with dialogue in Martin-Löf's intuitionistic type theory // Personal Computers & Local Network. Noviy Afon, 1986. P. 321–322.

Received for publication 5 November 2007