International Journal of Pure and Applied Mathematics

Volume 42 No. 4 2008, 469-474

PER MARTIN-LÖF'S TYPE THEORY FOR AUTOMATED PROGRAM WRITING

Gohar Marikyan

Empire State College
State University of New York
325 Hudson Street, New York, NY 10013, USA
e-mail: Gohar.Marikyan@esc.edu

Abstract: In [2] and [1] Per Martin-Löf has introduced his type theory. Any problem defined in the "language" of type theory is a type in that theory. To solve the problem, that is, to infer (deduce) the type that represents the problem in the theory is the same as to construct an object of that type. Based on the construction of that object, a program can be created that produces a solution of the problem. The automation of the above mentioned procedures will result in a system for automated program writing. To achieve this goal first the theoretical basis should be established. It requires a large number of scholastic proofs and results. The most important result is consistency of the theory. I have proved the consistency of Martin-Löf's type theory. In January 2006 it was submitted for publication in Springer Archive for Mathematical Logic.

In this article I describe Martin-Löf's type theory (MLTT) for automated program writing. MLTT is based on Martin Löf's type theory presented in [2] and [1].

AMS Subject Classification: 03A68
Key Words: Martin-Löf's type theory, automated program writing, automated problem solving

1. Introduction

In [2] Per Martin-Löf presented his type theory to be used as a base for a programming language. Subsequently, there were a few tries in that direction,

Received: August 17, 2007

however, without any positive outcome. My results show that his type theory is more valuable as a base for automated problem solving, and in this article I will describe the theory from the automated problem solving point of view.

Any mathematical object is an object of a certain type, and is always shown together with its type.

$$a \in A \tag{1}$$

means a is an object of type A. In this context there is no need to introduce a special term for proposition. Any proposition is a type whose objects are proofs of the proposition. Therefore, if A is a proposition then (1) means a is a proof of A. Likewise, if A is a problem then (1) means a is a solution of A. If A is a formula then the existence of a will mean that A is true.

A may have free occupancies of one or more variables. If so, then a may have free occupancies of the same variables.

A propositional function defined on the type A is called a property of objects of type A. If a is an object of type A, and B is a property of objects of type A, then $B(a)$ is the proposition that a has the property $B(a)$, and objects of $B(a)$ are the proofs that a has property B. $B(x)$ also can be viewed as a function. The domain of this function is the type A, and the range is a set of types.

Example. If N denotes the type of natural numbers, then $4 \in N$ means 4 is a natural number. If E denotes the property of being an even number, then the proposition that 4 is an even number is denoted by $E(4)$.

2. Expressions of MLTT

Expressions of MLTT are formed out of variables $x, y, z, \cdots$ by means of various forms of expression $(Fx_1, \cdots, x_n)(a_1, \cdots, a_m)$. Not necessarily all variables $x_1, \cdots, x_n$ become bound in all $a_1, \cdots, a_m$. Therefore, we always need to be clear about which variables are bound in which parts of an expression. A fully evaluated expression is called *canonical expression*. Expressions do not necessarily have a value. The list of all forms of canonical expressions are shown in [2].

3. Noncanonical Expressions of MLTT

All forms of noncanonical expressions of MLTT are listed below.

$c(a)$. To execute $c(a)$, we first execute c. If the value of c is $(\lambda x)b$, then we execute $b(a/x)$. The result is the value of $c(a)$.

Example. If c is a function, $c(a)$ is the value of c at point a, where a is from the domain of c.

$(Ex, y)(c, d)$. To execute $(Ex, y)(c, d)$, we first execute c. If the value of c is (a, b), then we execute $d(a, b/x, y)$. The result is the value of $(Ex, y)(c, d)$.

Example. $d(x, y)$ is slope of tangent of a given curve at point (x, y). To calculate the slope of the tangent at point c we first find value of c, that is, coordinates of the point c, then we substitute them for x and y in d.

$(Dx, y)(c, d, e)$. To execute $(Dx, y)(c, d, e)$, we first execute c. If the value of c is $i(a)$, then we execute $d(a/x)$. The result is the value of $(Dx, y)(c, d, e)$. If the value of c is $j(b)$, then we execute $e(b/y)$. The result is the value of $(Dx, y)(c, d, e)$.

Example. If c then execute d else execute e.

$J(c, d)$. To execute $J(c, d)$, we first execute c. If the value of c is r, then we execute d. The result is the value of $J(c, d)$.

Example. If $a = b$ then execute d.

$R_n(c, c_0, \cdots, c_{n-1})$. To execute $R_n(c, c_0, \cdots, c_{n-1})$, we first execute c. If the value of c is c_m for some $m = 0, \cdots, n-1$, then we execute c_m. The result is the value of $R_n(c, c_0, \cdots, c_{n-1})$.

Example. $R_n(c, c_0, \cdots, c_{n-1})$ corresponds to definition by cases, where c is the conditional expression that defines which of $c_0, \cdots, c_{n-1}$ should be executed.

$(Rx, y)(c, d, e)$. To execute $(Rx, y)(c, d, e)$, we first execute c. If the value of c is 0, then we execute d. The result is the value of $(Rx, y)(c, d, e)$. If the value of c is a', then we execute $e(a, (Rx, y)(a, d, e)/x, y)$. The result is the value of $(Rx, y)(c, d, e)$.

Example. Let us use $(Rx, y)(c, d, e)$ to calculate $2^{2^{2^2}}$. To calculate $2^{2^{2^2}}$ we have to execute $(Rx, y)(n, 2, 2^y)$ where n is the number of levels of exponents (in this case, $n = 3$). The value of $(Rx, y)(0, 2, 2^y)$ is 2. The value of $(Rx, y)(1, 2, 2^y)$ is $2^{(Rx,y)(0,2,2^y)}$. That is, 2^2. The value of $(Rx, y)(2, 2, 2^y)$ is $2^{(Rx,y)(1,2,2^y)}$. That is, 2^{2^2}. The value of $(Rx, y)(3, 2, 2^y)$ is $2^{(Rx,y)(2,2,2^y)}$. That is, $2^{2^{2^2}}$.

4. Operations on Types of MLTT

All operations on types of MLTT are listed below.

Cartesian Product of a Family of Types. If A is a type and B is a function which to an arbitrary object x of type A assigns a type $B(x)$, then cartesian product

$$(\prod x \in A)B(x) \tag{2}$$

is a type of functions that take an arbitrary object of type x of type A into an object of type $B(x)$. If $b \in (\prod x \in A)B(x)$ and $a \in A$, we may apply b to a and get $b(a) \in B(a)$. If $B(x)$ is a proposition, then (2) presents the proposition $(\forall x \in A)B(x)$. If $B(x)$ is the same for all x, then $(\prod x \in A)B(x)$ can be abbreviated to $A \to B$. If A and B are propositions, then $A \to B$ presents the proposition $A \supset B$.

Example. Let E be the type of all even natural numbers, and $M(x)$ be the type of all multiples of x, where $x \in E$. $3x$ is an object of $(\prod x \in E)M(x)$. To an arbitrary object of E, $3x$ corresponds an object of $M(x)$. $4 \in E$, then $3 \times 4 = 12$ is a multiple of 4, therefore, 12 is an object of $M(4)$. In the same manner $127x$, for instance, is another object of $(\prod x \in E)M(x)$.

Disjoint Union of a Family of Types. If A is a type and B is a function which to an arbitrary object x of type A assigns a type $B(x)$, then disjoint union

$$(\sum x \in A)B(x) \tag{3}$$

is a type of pairs (x, y) where x and y are objects of types A and $B(x)$, respectively. If $B(x)$ is a proposition, then (3) presents the proposition $(\exists x \in A)B(x)$. If $B(x)$ is the same for all x, then $(\sum x \in A)B(x)$ can be abbreviated to $A \times B$. If A and B are propositions, then $A \times B$ presents the proposition $A\&B$. A proof of $A\&B$ is a pair of proofs consisting of proofs of A and B.

Disjoint Union of Two Types. If A and B are types, so is the disjoint union

$$A + B \tag{4}$$

which is the type of objects of the form $i(x)$ with $x \in A$, or $j(y)$ with $y \in B$. If A and B are propositions, then $A + B$ presents the proposition $A \vee B$. A proof of $A + B$ is either a proof of A (with indication that A has been proven), or a proof of B (with indication that B has been proven).

Identity. If x and y are objects of type A, then

$$I(A, x, y) \tag{5}$$

is the proposition that x and y are identical. If $x \in A$, then $r(x)$ is the proof of the proposition $I(A, x, x)$.

Finite Types. For each non-negative integer n, N_n is type of non-negative integers preceding n with subscript n, that is, $N_n = \{0_n, 1_n, \cdots, (n-1)_n\}$. N_0

is an empty type to represent logical constant $False$ $\perp$. N_1 is used to represent logical constant $Truth$ $\top$. Negation is defined, as usual, by

$$\neg A = A \to \perp .$$

Natural Numbers. N is the type of natural numbers. 0 is an object of type N. If x is an object of type N, so is its succeeding x'.

Universes. $N_0, N_1, \cdots$ and N are objects of type V_0.

V_n is an object of type V_{n+1} where $n = 0, 1, \cdots$.

If A is an object of type V_n, B is a function which to an arbitrary object x of type A assigns an object of type V_m, then $(\prod x \in A)B(x)$ and $(\sum x \in A)B(x)$ are objects of $V_{max\{n,m\}}$.

If A is an object of type V_n and B is an object of type V_m, then $A + B$ is an object of type $V_{max\{n,m\}}$.

If A is an object of type V_n, x and y are objects of type A, then $I(A, x, y)$ is an object of type V_n.

5. Rules of Inference or Proof Rules of MLTT

The following are rules of inference or proof rules of MLTT. They can be found in [2].

The General Rules. Reflexivity, Symmetry, Transitivity, Equality of Types, and Substitution.

Rules of Cartesian Product of a Family of Types ($\prod$). $\prod$-formation, $\prod$-introduction, $\prod$-elimination, and $\prod$-equality.

Rules of Disjoint Union of a Family of Types ($\sum$). $\sum$-formation, $\sum$-introduction, $\sum$-elimination, and $\sum$-equality.

Rules of Disjoint Union of Two Types ($+$). $+$-formation, $+$-introduction, $+$-elimination, and $+$-equality.

Rules of Identity (I). I-formation, I-introduction, I-elimination, and I-equality.

Rules of Finite Types (N_n). N_n-formation, N_n-introduction, N_n-elimination, and N_n-equality.

Natural Numbers (N). N-formation, N-introduction, N-elimination, and N-equality.

Rules of Universes (V_n). V_n-formation, V_n-introduction, and V_n-elimi-

nation.

6. Conclusion

MLTT is a consistent powerful theory that can be used for automated problem solving and for creating software that automatically synthesizes computer programs.

References

[1] Per Martin-Löf, An intuitionistic theory of types: Predicative part, In: *Logic Colloquium '73* (Ed-s: H.E. Rose, J.C. Shepherdson), North–Holland, Amsterdam (1973), 73-118.

[2] Per Martin-Löf, Constructive mathematics and computer programming, In: *Sixth International Congress for Logic, Methodology, and Philosophy of Science*. North–Holland, Amsterdam (1982), 153-175.