AN INVESTIGATION ON THE LOGICAL STRUCTURE
OF MATHEMATICS (XIII)*⁾

A METHOD OF PROGRAMMING OF PROOFS IN MATHEMATICS FOR ELECTRONIC COMPUTING MACHINES

SIGEKATU KURODA

1. Preliminaries

In spite of invaluable importance of the fundamental circuit logic of performing the basic logical operators AND, OR, and NOT in achieving complicated *computations* for electronic computing machines, the research of planning and programming of *proving* a given mathematical or logical assertion for electronic computing machines is incomparatively far behind. This is of course partly because of little practical demand for solving such a theoretical problem, but there is also the essential reason for it that the proving procedures are much more difficult than the computing ones. Namely, the latter is achieved in attaining the *objective* (the result of computations) by the *method* which is given to the machine by program, while the former is only accomplished when the machine constructs a *method* by which the *objective* (the assertion to be proved) is attained. In proving procedures the machine has to look for the method of attaining a given objective, so that the direction of the action of the machine is completely reversed. What is to be given to the machine in proving procedures is not a definite way for attaining the objective, but the fundamental *rules* of constructing a proof, by the complicated combination of which the machine has to find a way, if any, to the objective that has been placed in the storage of the machine.

We do not enter here into actual programming but we note that detailed research

2. Fundamental rules of proofs

As far as the basic logical operators AND, OR, and NOT, already mentioned above, are concerned, the rules to be given to the machine as the basis of a proof are very simple as in the case of the fundamental circuit logic. Namely,

(1) From F AND G to F ; G

(2) From NOT (F AND G) to NOT F or NOT G

(3) From F OR G to F or G

(4) From NOT (F OR G) to NOT F ; NOT G

where F, as well G, denotes a juxtaposition of symbols, which we shall later define and call "formula".

Besides these, we need the following four rules concerning the so-called quantifiers, universal and existential, of logic.

(5) From   FOR ALL F^x to F^w

(6) From NOT (FOR ALL F^x) to NOT F^m

(7) From   FOR SOME F^x to F^m

(8) From NOT (FOR SOME F^x) to NOT F^w

where w and m are variables and, in particular, w in (5) and (8) is called the

---

of programming along this line is successfully going on by several mathematicians, using the computing machine M-1 in the Electrical Communication Laboratory in Tokyo and TAC in the Faculty of Engineering, University of Tokyo.

Meanwhile, we have come to be accessible to the publications at the International Conference on Information Processing, Paris, 1959, in which several researches on proving mathematical theorems by electronic computing machines are reported. See H. Gelernter: Realization of a geometry theorem proving machine, UNESCO/NS/ICIP/ 1.6.6; A. Newell, J. C. Shaw, and H. A. Simon: Report on a general problem-solving program, UNESCO/NS/ICIP/1.6.8; B. Dunham, R. Fridshal, and G. L. Sward: A non-heuristic program for proving elementary logical theorems, UNESCO/NS/ICIP/1.6.10; P. C. Gilmore: A program for the production of proofs for theorems derivable within the first order predicate culculus from axioms UNESCO/NS/ICIP/6.1.14. See also H. Wang: Toward Mechanical Mathematics, reported in Gelernter's paper, cited above, to be published in IBM Journal of Research and Development.

In programming of mathematical proofs it is especially necessary to apply heuristic method or strategy (see H. Gelernter's paper cited above) in order to minimanize the numbers of mechanical trials.   The method of using UL has the advantage for this purpose because UL is faithful to the "meaning" of concept formation in mathematics and is eligible for mechanization of proofs (see also § 5, mechanization of mathematics, Part (IV), this J. vol. 13).

eigen variable of the rules (5) or (8), respectively, and F^x, F^w, and F^m are again formulas containing the free variable explicitly written. The rules of use of eigen variables and also the definition of a variable are given later.

These eight rules are all the fundamental rules we need in proving a given mathematical or logical assertion. By virtue of the duality between AND and OR and between ALL and SOME, the above eight rules are reduced to four rules e.g. (1), (2), (5), and (6).

Any complicated proof in mathematics is nothing else than a complicated superposition of these four fundamental rules, just were any computer function interconnections of the fundamental circuit logic. Naturally there is another rule about how to use these fundamental rules in a mathematical proof. Before explaining these rules we have to explain what is a mathematical assertion.

3. Symbols

In order to describe any mathematical assertion, besides the symbols already mentioned, namely, $\not>$ (NOT), $\land$ (AND), $\lor$ (OR), $\forall$ (FOR ALL), $\exists$ (FOR SOME), and variables such as $a, b, \ldots m, n, \ldots x, y, \ldots w, \ldots$, we need *only one more symbol* $\in$. We use also the logical operators $F \to G$, $F \equiv G$ as abbreviations of $\not> F \lor G$, $(F \to G) \land (G \to F)$ respectively, and the equality $a = b$ as the abbreviation of

$$\forall x. \ x \in a \equiv x \in b.$$

By using these symbols we construct *formulas* as follows.

4. Variables and formulas

Definition of formulas of order 0.

Assume that we have an indefinite number of variables $a, b, \ldots x, y, \ldots$ which can be used as *independent variables* at our disposal.

(i) $a \in b$ is a formula where a and b are independent variables.

(ii) If F and G are formulas then $F \land G$ and $F \lor G$ are formulas.

(iii) If F is a formula and the negation is not the outermost logical operator of F, then $\not> F$ is a formula.

(iv) If F^x is a formula where x is an *independent* variable, free and not bound in F^x, then $\forall x F^x$ and $\exists x F^x$ are formulas.

An independent variable x occurring in a formula F is called *free* or *bound*

in F, according as there is an x in F which does not or does lie under the operation scope of $\forall x$ of $\exists x$ in F respectively. An independent variable in a formula can be free and bound at the same time.

Definition of dependent variables of order 0.

Assume that we have an indefinite number of variables p, q, . . . , which can be used as *dependent variables* at our disposal.

(v) If F^u is a formula of order 0 where u is an independent variable, free and not bound in F^u, then a new *dependent variable* p of order 0 can be introduced with

$$(pD) \qquad\qquad \forall x_1 \cdots x_n \forall u. \; u \in p \equiv F^{u;\,x_1,\,\cdots,\,x_n}$$

as its *defining formula*, where u, x_1, . . . , $x_n (n \geq 0)$ are the complete system of free variables in F^u. F^u is called the *definiens* of p and u the *element variable* of p. The variable p is called to depend on x_1, . . . , x_n and p is written also as $p^{x_1,\,\cdots,\,x_n}$. The independent variables upon which a dependent variable q depends are called free in the formulas $q \in k$ and $k \in q$.

Definition of formulas and dependent variables of order $m (\geq 1)$.

(vi) Assume that the formulas and dependent variables of order $m - 1 (m \geq 1)$ have been defined. If, in (i) above, a and b represent any independent variables or dependent variables of order less than m, then (i), (ii), (iii), and (iv) give the definition of formulas of order m. If F^u in (v) is a formula of order less than m, then (v) gives the definition of the dependent variable p of order m.

Formulas and dependent variables of any order are simply called *formulas* and *dependent variables* respectively.

If no independent variable other than u is free in the definiens of F^u of a dependent variable p, p is called a *constant*.

Definition of subordination and superordination of dependent variables.

A dependent variable q which occurs in the definiens of F^u of a dependent variable p is called (directly) *subordinate* to p. A dependent variable subordinate to a dependent variable which itself is subordinate to p is called subordinate to p. If a dependent variable q is subordinate to p, then p is called *superordinate* to q.

5. Assertions and proofs

Definition of an assertion

We denote by (I) the formula expressing the *principle of extensionality:*

(I) $$\forall xyz. \ x = y \wedge x \in z \rightarrow y \in z.$$

Let σ be a sequence of a finite number of defining formulas, including possibly the formula (I). If for any dependent variable p defined in σ, any dependent variable subordinate to q is defined in σ, then σ is called *closed.* Assume that σ is closed, and let H be a formula in which no dependent variables occur other than defined in σ. Then we call

(A) $$\sigma \vdash H$$

is an *assertion*, σ the premises of (A), and H the *conclusion* of (A).

Definition of a proof

We shall give the definition of a proof of a given assertion (A). To do this, we define first the *association* of a *proof constituent* with a proof formula. These are the rules (1)-(8) stated before. Namely:

We associate the proof consituent

(i)	$\overline{F \ \ G}$	with the formula	$F \wedge G$
(ii)	$\overline{\nearrow F}$ or $\nearrow G$	"	$\nearrow . \ F \wedge G$
(iii)	$\overline{F}$ or G	"	$F \vee G$
(iv)	$\nearrow F \ \ \nearrow G$	"	$\nearrow . \ F \vee G$
(v)	F^w	"	$\forall x F^x$
(vi)	$\overline{\nearrow F^m}$	"	$\nearrow \forall x F^x$
(vii)	$\overline{F^m}$	"	$\exists x F^x$
(viii)	$\overline{\nearrow F^w}$	"	$\nearrow \exists x F^x$

where m is any variable, dependent or independent, and w is an independent variable which is called, as is mentioned before, the *eigen variable* of the proof constituent concerned.

When C is any formula we use the proof constituent

$$C \ \ \nearrow C$$

which is called a *cut* with C and $\nearrow C$ as *cut formulas.*

Now, we define a proof of the assertion (A). A *proof* P of (A) is a

(reversed) tree form superposition of proof constituents such as

$$\nwarrow\!\sigma, H$$

which has the four properties, stated below.   Here $\nwarrow\!\sigma$ is the sequence consisting of the negation of each formula of σ. $\nwarrow\!\sigma, H$ is called the *top-sequence* of P and a formula belonging to the top sequence $\nwarrow\!\sigma, H$ of P a *top formula* of P.   A P-formula under which there is no horizontal line is called a *bottom formula* of P.   A P-*string* is a way, starting at a bottom P-formula and ending at the P-top sequence, running through each formula which is upon each horizontal line, touched on the way.   All the formulas of P-top sequence are considered to belong to the string.

(i) *Association property*: For any P-constituent E there is over E a formula with which E is associated, unless E is a cut.

(ii) *Cancelling property*: Any P-string contains a pair of formulas, the one of which is the negation of the other.

(iii) *Independent variable restriction*: The eigen variable of a P-constituent E does not occur free in P over E, except in $\nwarrow\!\sigma$.

(iv) *Dependent variable restriction*: Any dependent variable occurring in P is defined in σ.

This is the definition of a proof of the assertion (A).

Restriction of variables

For the necessity of formulating a proof in mathematics more adequately, we shall generalize the definitions of an assertion and a proof, given above, as follows.   Let $x_1, \ldots, x_n$ be the complete system of independent variables in the definiens F^u of the defining formula (Dp) of p, then, instead of (Dp), we use the formula

$$(\mathrm{D}^*p) \qquad \overset{m_1}{\forall} x_1 \cdots \overset{m_n}{\forall} x_n \overset{k}{\forall} u.\ u \in p^{x_1, \cdots, x_n} \equiv F^{u;x_1, \quad ,x_n},$$

where $m_1, \ldots, m_n, k$ are any independent variables or dependent variables previously defined, and the notation such as $\overset{m}{\forall} x$ (and $\overset{m}{\exists} x$) means the restriction of the scope of the bound variable x to m, i.e. $\overset{m}{\forall} x$ (and $\overset{m}{\exists} x$) is the abbreviation

of $\forall x.\ x \in m \rightarrow$ (and $\exists x.\ x \in m \wedge$).   We call the formula $(\mathrm{D}^* p)$ the *defining formula of p with restriction of variables*, in which some or all of $x_1, \ldots, x_n$ and u may be left universal, that is, some or all of $m_1, \ldots, m_n$ and k may be lacking.

We generalize the definition of an assertion $\sigma \vdash H$ in allowing that the formula $(\mathrm{D}^* p)$ may be used instead of the defining formula $(\mathrm{D}p)$ of p occurring in σ.   The definition of closedness of the premises σ should be affected accordingly so that the defining formulas of the dependent variables, such as $m_1, \ldots, m_n, k$ used for restriction of the ranges, should also be included in the closed premises σ again possibly in the form with restriction of variables. The definition of a proof for an assertion $\sigma \vdash H$ in the generalized sense is the same as that given above.

6. Universality of logic and machines

It is usually said that a computing machine should be or is of universal character.   What it means is, precisely speaking, nothing more nor less than the fact that the machine can perform what the machine can, so that the concept of universality should rather be regarded as defined by the machine itself, unless we have a precise definition of universality, independent of the machine. So, for instance, the concept of a computable function was defined by Turing as the function the value of which is obtained from any given arguments of the function by the theoretical machine now we call by his name.   However, if one could let a machine perform, or had the insight of being able to let it perform, "any" mathematically rigorously formulated "algorithm", then one would come to embrace the "thesis" that the machine is universal.

On the other hand, it seems very presumable that what we call usually a proof in mathematics can be formulated as a proof defined above.   So we define as UL (Universal Logic) the logical system in which the variables, formulas, and proofs are defined as above, and then we can consider a method of programming of a UL-proof of a given UL-assertion for an existing computing machine.

It is particularly to be noted that the cardinal problem of the foundational research about consistency and inconsistency is put quite aside from our present concern.

7. Undecidability and proofs

There is in logic a series of results, obtained by Gödel, Church, Quine, etc., which prove the impossibility of some attempts formulated precisely. To these results belong the non-existence of decision procedures for all formulas in predicate logic even with a binary relation (Church, Quine), and the existence of undecidable statements in a sort of axiomatic theories in mathematics, satisfying certain conditions (Gödel).

Therefore, there is no mechanical procedures to prove all the "true" dependent-variable-free UL-assertions $\vdash H$, and there is a UL-assertion $\sigma \vdash K$ in a certain axiomatic theory T, for which both $\sigma \vdash K$ and $\sigma \vdash \diagup K$ are T-unprovable. In such a situation it seems meaningless to endeavour to construct a proof of $\sigma \vdash K$, or $\sigma \vdash \diagup K$. However, it may happen that τ, $\sigma \vdash K$ or ρ, $\sigma \vdash \diagup K$ is provable for some appropriate premises τ, σ or ρ, σ.

As is stated above, it is proved by Gödel that there is an undecidable statement if we fix some axiomatic theory. This does not mean at all that for a given assertion there is no consistent axiomatic theory in which the assertion is decidable. Gödel's theorem asserts the weakness of a fixed axiomatic theory but not the weakness of mathematics. It is the mathematician's idea to determine an enough number of appropriate premises in order to prove some given assertions. It is, therefore, not meaningless to look for a proof of $\sigma \vdash H$, if one has the confidence that the premise σ is enough to prove the conclusion H. Even in such a case, the length of a possible proof of $\sigma \vdash H$ is unbounded. We mean by the *length* of a proof the maximum of the numbers of horizontal lines which each string of the proof crosses. It is proved that a proof of $\sigma \vdash H$ of less than a given length can be determined by a finite number of procedures and such a proof, if any, can be constructed by a machine in principle, unless the proof does not exceed the capacity of the machine.

However, a tremendous number of trials is necessary, if we wish to construct a proof mechanically. The number of trials may be reduced to a smaller number both by using the properties of proofs and by making use of the device of the machine. Herein lies the fundamental problem of programming of mathematical proofs.

It is unbelievable that an existing computing machine can perform so elaborated a proof in mathematics. But it is doubtless that it can perform

some simpler proofs in mathematics by a suitable programming.   It is also a problem to examine whether a computing machine is provided with a qualified potency as *proving machine*, since a special kind of procedures is strongly required in proving process, but not all kinds of procedures needed in computing procedures.   It is *equally important what machine is most suitable for* the purpose of performing mathematical proofs and what formulation of mathematics is most convenient for the same purpose.   The logical system UL seems to be one of the most useful formulations of mathematics for this purpose and UL is, as is well known, translated into part of quantification theory (pure predicate logic).

(Added in proof, February 29, 1960) H. Wang's paper, mentioned in foot-note*), appeared in IBM Journal of Research and Development, vol. 4, no. 1, pp 2–22, January 1960.   Wang reports in it that he wrote three programs on an IBM 704, by the first of which all the theorems, amounting to over 200 in number, in the first five chapters of *Principia Mathematica* were proved in less than 3 minutes by IBM 704.   He obtained also many other interesting results of having the machine prove formulas of predicate logic.

Independently of Wang's research, two different programs of proving mathematical theorems have been written since last autumn, one by T. Simauti *and others on TAC (see foot-note*)) and the other by S. Takasu on M-1 (ibid.).* These programs have recently been finished and proofs of some theorems were performed by the machines. Namely: TAC proved, for instance, on February 11, 1960, the equality of left and right identities of an algebraic system:

$$\forall x.\ xe = x.\ \wedge \forall x.\ e'x = x.\ \wedge \forall xyzw.\ xy = z \wedge xy = w \to z = w.\ \to e = e'$$

in about 61 sec., not including input and output time; M-1 proved, for instance, on February 29, 1960, the formula $\vdash * 2$ (see Nagoya Math. J. vol. 13, p. 41):

$$a \subseteq b \to \sigma \vdash a \subseteq {}_\sigma \vdash b$$

in about $90 \sim 120$ sec.   These results, including the description of the programs and the feature of TAC and M-1, will be published elsewhere.

Mathematical Institute
Nagoya University