COMPUTER UNDERSTANDING OF MATHEMATICAL PROOFS

Vesko Marinov*

Institute for Mathematical Studies in the Social Sciences
Stanford University
Stanford, California 94305

Abstract

Mathematical proofs constitute a mixture of formulas with a subset of natural language. They can be represented as a sequence of lines expressible in the symbolism of predicate calculus. The transition from step to step may depend on a series of logical manipulations and/or on intricate mathematical knowledge associated with the domain of the proof. The organization of the proof may depend on different conventions adopted by mathematicians in communication with each other. This paper deals with problems involved in following the mathematical argument along those lines. Some of the ideas were implemented as a part of a system for teaching axiomatic set theory to college students. The most powerful and frequently used rules of inference utilize a resolution theorem prover. To the best of our knowledge this is the only resolution theorem prover, perhaps the only general purpose theorem prover used in actual production.

Key Words

Proof understanding, automatic proof checking, automatic theorem proving, computer assisted instruction.

Introduction

A mathematician writes down a proof of a theorem on a piece of paper and hands it to a colleague. The latter reads through it, usually understands it, and all too frequently finds an error, which means that at least in its present form it is not a proof at all. Sometimes it takes several readers until an existing error is detected and on occasion it has taken many years to discover errors in some difficult proofs. It would be very useful if our mathematician could type in the computer whatever he wrote and receive some for of the response he received from the human reader. The paper examines problems encountered during initial efforts toward such a goal.

Furthermore, if one attempts to teach higher-level mathematics by computer, as it is currently being done for set theory at Stanford, it is essential that the machine understands the proofs of the students to the extent that it accepts only the correct ones. The paper will discuss some of the problems and experiences with the proof checker which is the most important part of the system for teaching axiomatic set theory (named QUIP) developed at the Institute for Mathematical Studies in the Social Sciences (IMSSS) [6].

The CAI system has been used since the fall quarter of 1974 to teach Philosophy 161, "Introduction to Set Theory", at Stanford University. The program was written in LISP and SAIL and runs on the TENEX operating system for the DEC PDP-10 computer. The course is for upper-level undergraduate students and presents axiomatic set theory as developed in [9]. The examples here are from proofs of theorems from the curriculum of the course. All illustrations of the ideas embodied in the mechanism for handling the proofs are as of the time the author left IMSSS. There have been some improvements since.

This paper will focus on the issues involved in understanding mathematical proofs. To improve his understanding of the examples a reader unfamiliar with proof checking should consult Suppes [8] where the logical system underlining the proof checker of QUIP is explained. Furthermore, due to lack of space an explanation of the organization, features and performance of the entire CAI system cannot be included. It is available in [6].

Natural Language Part

There are two major problems facing computer understanding of mathematical proofs. First, the language of mathematics is not that of predicate calculus. It usually is a mixture of formulas with natural language. Thus, one of the problems theoretically approximates that of understanding natural language. In fact, this is not so, as some preliminary results on the language of sets [6] indicate that the language of mathematics, being a very restricted subset of natural language, will probably be handled much more easily.

A mathematical argument can be represented as a sequence of separate steps. Our opinion is that the best internal representation of the formulas describing a step in a proof is predicate calculus. In this view the main effort of the natural lan-

Present address: Department of Computer Science, Oregon State University

guage processing will be spent in correct translation of the steps in the argument, although it will also affect the structure of the proof and the choice of how to go about making a step in the proof. The remaining problem, to which this paper is devoted, is how to follow the logical steps in the proof.

Formal and Informal Proofs

In communications among mathematicians or between a teacher of mathematics and his students the steps in the proofs are as large as it is optimal for understanding. They are large enough to avoid obvious but tedious details which would usually make the proof less clear. On the other hand, they are small enough to make it possible for the reader to follow the proof without having to discover it himself. The size of the steps varies widely with different proofs and in some cases only a very general outline is given, thus requiring a greater effort on the part of the reader. In any event, that is what we call an informal proof. In addition, the concepts of being essential, unfamiliar, or trivial in a proof are not precise at all. This very vagueness of the criteria governing informal proofs is a primary justification for a precise definition of a formal proof [8].

Formal proofs require precise rules for justifying each step. In that respect they are somewhat algorithmic in character. It is no wonder that automatic proof checking is almost as old as artificial intelligence itself [5]. One of the first computer proof checkers was that of Abrahams [1], which implemented an extension of the logical system of Suppes [8]. IMSSS at Stanford has offered a CAI course in symbolic logic using a proof checker based on Suppes' system since 1963. A later implementation is described in [2].

Size of the Steps

The problem with such formal proofs is that their steps are quite small. A proof of somewhat greater difficulty in set theory (or other mathematical theory) could easily go up into several hundred steps. Such proofs cannot be used for teaching axiomatic mathematics and hardly for any other purpose. This problem has been resolved to a large degree in the proof checker of QUIP.

Before discussing the techniques used in accepting larger steps we shall illustrate some of the points made above with a comparative example. First is the proof of Theorem 53 on page 112 in [9], Then follows the proof of the same theorem as it was accepted by QUIP. K(A) is the cardinal number of a set A, while ≈ means equipollent. Axioms, definitions, and theorems referenced in example proofs are given in the appendix. The command to the proof checker precedes each line. Some of the commands may seem long but are not entirely typed by the user, since the system has a recognition feature (see [6]). The QUIP proof below is presented as listed by the command "Review". In doing the first three steps a bit of additional typing is required where the user indicates that he wants to specify variables upon which the system asks him what values to assign. In the last two steps the lines have to be typed after the command.

A proof from "Axiomatic Set Theory" by P. Suppes:

Theorem 53. There are sets A and B such that (i) $A \cap B = 0$, (ii) $K(A) = m$, (iii) $K(B) = n$.

Proof. In view of Definition 7 (Cardinal Number), there are sets A1 and B1 such that $K(A1) = m$ and $K(B1) = n$, and by virtue of Theorem 9 there are sets A and B such that $A \cap B = 0$, $A \approx A1$, and $B \approx B1$. By the axiom for cardinals, then, $K(A) = m$ and $K(B) = n$. Q.E.D.

Proof of the same theorem as accepted by the proof checker:

Theorem 5.1.1

Derive: $(\exists A,B)(A \cap B = 0 \wedge K(A) = m \wedge K(B) = n)$

Df. of Cardinal Number
(1) $K(A1) = m$
Df. of Cardinal Number
(2) $K(B1) = n$
Theorem 3.8.9
(3) $A1 \approx A \wedge B1 \approx B \wedge A \cap B = 0$
1, 2, 3, Axiom for Cardinals VERIFY
(4) $K(A) = m \wedge K(B) = n$
3, 4 VERIFY
(5) $(\exists A,B)(A \cap B = 0 \wedge K(A) = m \wedge K(B)) = n$
Q.E.D.

Notice that the steps of the QUIP proof follow exactly the informal proof from the book. In the first three steps a definition and a theorem are invoked as the appropriate instantiations are made. The system has sorted variables and the letters m and n are among those used for cardinals. This enables us for example to get Line 1 directly from the definition of cardinal number. The QUIP proof requires Line 5 to obtain a formula which is a variant in terms of bound variables (in this case identical) to the statement of the theorem. The informal proof quits at Step 4. This is one of the many shortcuts used in informal proofs, some of which will be discussed below. They will eventually present the greatest problem in accepting informal text proofs and will require a large amount of additional information to be stored with the internal representation of a line.

Step 4 is the most interesting in this proof. It is achieved by a call to a resolution theorem prover giving it the lines 1, 2, and 3 plus the axiom for cardinals as references. Below is the continuation of the proof in the formal system of Suppes [8] as implemented in [2]. Preceding each line to the left are the rules of inference as typed by the user. In the brackets to the right their names are expanded.

Axiom for Cardinals
(4) $K(A1) = K(A) \leftrightarrow A1 \approx A$
3 LC [Left Conjunct]
(5) $A1 \approx A$

```
4,5  RQ          [Replace Equivalence]
     (6)  K(A1) = K(A)
1,6  RE          [Replace Equals]
     (7)  K(A) = m
Axiom for Cardinals
     (8)  K(B1) = K(B)  ↔  B1 ≈ B
3    RC          [Right Conjunct]
     (9)  B1 ≈ B ∧ A ∧ B  = 0
9    LC          [Left Conjunct]
     (10) B1 ≈ B
9,10 RQ          [Replace Equivalence]
     (11) K(B1) = K(B)
2,11 RE          [Replace Equals]
     (12) K(B) = n
7,12 FC          [Form Conjunction]
     (13) K(A) = m ∧ K(B) = n
```

As shown above, Step 4 requires 10 elementary steps. Yet the informal proof only says it follows "by the axiom for cardinals". For the mathematician this is an "obvious" step. In [9] one frequently finds lines in proofs that follow by sentential logic (p. 29, p. 52) or by quantifier logic (p. 45, p.60). When a step is justified by sentential logic it is decidable and one can safely lean on a tautology checker. It is our belief that the most efficient mechanical way to confirm a tautology is to check the truth table and thus QUIP has a TAUTOLOGY rule based on that principle. Steps justified by logical manipulations involving quantifiers are undecidable in general and consequently represent a much harder problem to be verified mechanically. For this purpose we employ a resolution theorem prover with equality replacement.

Resolution Theorem Prover in Action

The experience with QUIP shows that a well-organized resolution theorem prover gets most of the inferences seen intuitively by the user while working on a proof. This is not quite true for proofs involving equality, where some improvements to the prover are needed. The user has no interaction with the theorem prover except for asking a formula to be verified and supplying the references, from which he thinks the formula follows. An idea about the power of the theorem prover can be gained from the following theorems from the chapter on finite and infinite sets in QUIP's curriculum, which were proved in one step.

```
Theorem 4.1.24  Finite(A) ∧ B⊂A → B≺A
Follows from:
1) Df. of Proper Subset
   (∀A,B)(A⊂B ↔ A⊆B∧A≠B)
2) Th. 3.9.2  (∀A,B)(A  B → A  B)
3) Th. 3.9.16 (∀A,B)(A  B ↔ A  B  A  B)
4) Th. 4.1.23 (∀A)(Finite (A) → Dfinite(A))
5) Df. of Dedekind finite
   (∀A)(Dfinite (A) ↔ ¬(∃C)(C⊂A∧C≈A)
```

The above theorem was proved in 7.4 seconds of CPU time.

```
Theorem 4.3.2
(∃A,B)(Infinite(A) ∧ Infinite(B) ∧ A ⋠ B)
Follows from:
1) Df. of Infinite Set
   (∀A)(Infinite (A) ↔ ¬ Finite (A))
2) Th. 4.3.1  (∃A) Infinite (A)
3) Th. 3.9.17 (∀A) A≺P(A)
4) Th. 3.9.13 (∀A,B)(A≼B → B ⊀ A)
5) Th. 3.9.1  (∀A,B)(A≈B → A ≼ B)
6) Th. 3.8.2  (∀A,B)(A≈B → B ≈ A)
7) Th. 4.1.17 (∀A)(Finite(P(A))→Finite(A))
```

It was proved in 3.7 seconds of CPU times.

```
Theorem 4.1.22 Finite(A) ∧ ¬Finite(B) → A≺B
Follows from:
1) Th. 3.9.16 (∀A,B)(A≼B ↔ A≈B ∨ A≺B)
2) Th. 4.1.19
   (∀A,B)(Finite (A) ∧ A≈B → Finite (B))
3) Th. 4.1.20
   (∀A,B)(Finite (A) ∧ B≺A → Finite (B))
4) Th. 4.1.21
   (∀A,B)(Finite (A) → A≺B ∨ A≈B ∨ B≺A)
```

A student proof of the last theorem usually looks more like the following:

```
Derive:  Finite (A) ∧ ¬Finite (B) → A≺B
Hypothesis
    (1)  Finite (A) ∧ ¬Finite (B)
1C1, Th. 4.1.21  IMPLIES [1C1:  Line 1,
                              Conjunct 1]
    (2)  A≺B ∨ A≈B ∨ B≺A
Assume
    (3)  A≈B
1,3, Th. 4.1.19  CONTRADICTION
    (4)  A≠B
Assume
    (5)  B≺A
5, Th. 3.9.16  VERIFY
    (6)  B≼A
1,6, Th. 4.1.20  CONTRADICTION
    (7)  B⊀A
2,4,7 TAUTOLOGY
    (8)  A≺B
1,8 Conditional Proof
    (9)  Finite (A) ∧ ¬Finite (B) → A≺B
Q.E.D
```

This proof should be transparent even to those unfamiliar with Suppes' logical system [8]. The theorem can be proved in one step, yet the user usually sees the exact references on which this step depends only after developing the proof in the latter form. Furthermore, this form is more like the way mathematicians prove theorems. Still notice that the resolution prover was called upon for Steps 4, 6, and 7.

The prover is used primarily for the rules VERIFY and CONTRADICTION. While using VERIFY the user has to type the line, whose negation together with the references is passed to the prover. If the prover is able to confirm the inference it signals the proof checker to accept the line. For the CONTRADICTION rule the user merely points to the references which he believes form an inconsistency. If such is detected by the prover, the proof checker returns the negation of the last assumption on which the references depend. (Presumably there must be an incorrect assumption in order to reach a contradiction).

Implementation and Strategy

The main reason for selecting a resolution theorem prover was our belief that for the same generality and the same power it can be designed in a much more compact way than a heuristic theorem prover. For the purposes we are using it, simply a mechanical tool is needed and resolution seems to be exactly that. Later we shall discuss the possibility of a heuristic coupler to the proof checker, which would make the prover serve better the needs of understanding informal proofs. The prover was written in UCI-LISP. Together with the converter of the formulas into clausal form it is about 10 pages of pretty-printed code.

The prover employs the MU strategy. It consists mainly of keeping only resolvents containing merge literals or having a unit parent. It is shown in [3] that if in a refutation there are resolvents not satisfying the above restriction, there always exists another refutation from the same input set where such resolvents are obtained first. With this in view the strategy occasionally allows for a round of general resolution after which the restriction is imposed.

One thing that has plagued work on resolution in the past has been preoccupation with completeness. Recognizing that a prover is working in an undecidable domain it is obvious that completeness is going to be restricted by the real factors of time and space. The main objective in choosing a strategy and tuning a prover's parameters is optimizing the number of inferences it gets. It is the author's conviction that in this context incompleteness is a feature, rather than a drawback. Thus, completeness in our prover is restricted severely in many different ways.

Experiments with different strategies for resolution, carried out earlier by the author at the University of Texas have shown the MU strategy to be quite efficient in the set-theoretical domain. One property of the MU strategy, coupled with a limit on the depth of functional nesting in the resolvents, is that it usually runs quickly out of possibilities to resolve when given a satisfiable set of clauses (i.e., insufficient references). This is very important because one very frequent error of the student users has been to supply insufficient or incorrect references. In such a case it is very desirable that the prover detects this fact as soon as possible, rather than grind until the time limit is reached. This property has strongly influenced the selection of the MU strategy.

Deficiencies

Probably the most frustrating property of the prover from a CAI point of view has been the fact that thus far it has been impossible to characterize the class of theorems accepted by the prover despite substantial effort on the part of members of the IMSSS staff. This is most likely a consequence of the unnatural way in which resolution works. Sometimes the prover is able to verify steps much larger than the user can see, while other times it fails at steps which the user expects to be accepted. It would have been nice if one could give the users some more accurate idea what to expect from the prover.

Inferences which are missed by the prover, while being obvious to the user, are largely due to the fact that resolution breaks down the formulas to the atomic level before it can find a proof. For example, the inference

$$(\forall m,n,p,A,B)\{A\cap B=0 \wedge K(A)=m \wedge K(B)=n \wedge K(A\cup B)=p \rightarrow m+n=p\}$$

from

$$(\forall m,n,p)\{(\exists A,B)[A\cap B=0 \wedge K(A)=m \wedge K(B)=n \wedge K(A\cup B)=p] \leftrightarrow m+n=p\}$$

cannot be verified by the prover. The human user, though, with his well-developed abstraction capabilities quickly sees that the left side of the implication can be looked at as ø(m,n,p,A,B) . Hence the inference becomes:

$$(\forall m,n,p,A,B)[\Phi(m,n,p,a,B) \rightarrow m+n=p]$$

from

$$(\forall m,n,p)[(\exists A,B)\Phi(m,n,p,A,B) \leftrightarrow m+n=p].$$

Of course, seen this way, the inference is immediate for the prover, too. There is a provision in the program for the user to pass to the prover the abstracted form of the formulas. What would be desirable here is to have a heuristic coupler between the prover and the proof checker which looks at the possibilities of abstraction in the set of formulas.

Implication Rule

Proofs in axiomatic systems frequently include transformations of formulas justified by the application of definitions, axioms, or theorems. Almost always these inferences include quantifiers and would fall within the domain of the theorem prover. However, such inferences are deterministic and one potentially loses power in taking a chance with the prover. Many definitions are by equivalence, which is a simple connective for the human, but the worst one for resolution. Furthermore, for the rules VERIFY and TAUTOLOGY the formula must be typed in, which the users are not very enthusiastic about.

All these considerations led to the development of the IMPLIES rule. Variables bound by an universal quantifier containing the main connective of the justifying formula within its scope are matched to any terms in the formula to be transformed. For example, let

(1) f"C $\subseteq$ f'D

be a line in a proof (f"C means the image of the set C under the function f). From it we can infer the line

(2) $(\forall x)(x \in f''C \rightarrow x \in f''D)$

just by referring to the definition of subset:

$$(\forall A,B)(A\subseteq B \leftrightarrow (\forall x)(x \in A \rightarrow x \in B)).$$

In a case like this, which happens most fre-

quently, the rule is equivalent to universal specification and modus ponens. The rule also allows for (1) to be inferred from (2) by the definition of subset. In some cases the matching is much more intricate than in the example above. The matching algorithm utilizes a subset of unification. The same algorithm is used in matching terms for replacements by equality in the proof. More details can be found in [4].

Matching of Schemas

A more sophisticated application of the matching algorithm is the matching of theorem schemas. Sometimes axioms and theorems contain formula schemas which can be matched to any formula satisfying some variable constraints. For example, the theorem

(3) $(\exists B)(\forall x)(x \in B \leftrightarrow FM(x)) \rightarrow (\forall y)(y \in \{x:FM(x)\} \leftrightarrow FM(y))$

will match the proof line

(4) $(\exists C)(\forall z)(z \in C \leftrightarrow z \subseteq D)$

and return

(5) $(\forall y)(y \in \{x:x \subseteq D\} \leftrightarrow y \subseteq D)$.

Here, FM is matched to $z \subseteq D$, while the match between the designated variable x of the schema FM(x) and z is carried along.

Occasionally it is not straightforward to determine which variable to match with the designated variable of the schema. Then one has to look at which quantifier matched the innermost quantifier over the designated variable. Since the latter is always quantified, the algorithm would succeed if there is a possible match. Then, when the rule is returning, the current designated variable is substituted for the value of the variable of the same schema when the match was first made. (Of course, any subsequent conflict would have failed the match). Thus at the point of recovering the value of FM(y) from (3), y is substituted for z in $z \subseteq D$ (the match to FM) and $y \subseteq D$ is returned as seen in (5). Recall that x was associated with z when $z \subseteq D$ was matched to FM(x) and now y corresponds to x.

Embodiment of Mathematical Knowledge

Some lines in proofs are produced by mathematicians without references to the justifying definitions or theorems. They usually express well-understood and well-organized domains of knowledge shared by the workers in a particular field. Exact references in such cases would obstruct rather than facilitate communication. A very simple example from set theory is the line

(6) ASC A B£C -> AVBSC.

To justify this line one should really refer to the definitions of subset and union, although no student of set theory does so once he has grasped the Boolean operations and relations between sets. More complicated examples could easily be found in proofs on set theory, for example, in treating functions, Cartesian products, etc.

It would be desirable if "bags" of such knowledge were embodied in decision methods. A good example is the BOOLE rule, which is a part of QUIP [7], This rule uses a decision method by Quine and converts the Boolean operations and relations between sets into propositional connectives, after which a tautology check is applied.

This rule could possibly be extended to cover more of the set-theoretical knowledge, and other similar rules could be developed. However, it is not clear how much of set theory could be embodied in decision methods. There certainly will be need for some heuristic knowledge, organized as well as possible. In fact, such embodiments would be a very important part of any completed system handling some domain of mathematics.

Goal Hierarchy

Informal proofs employ many conventions, assumptions, and shortcuts not explicitly mentioned in the proof. They largely depend on the domain of the theory in which the theorem is being proven, although there are many conventions which apply generally to mathematics. For example, it is customary when one wants to show

(7) $A \subseteq B$

to assume

(8) $x \in A$

and eventually derive

(9) $x \in B$.

Thus the proof (or part of proof) is terminated, but the implicit assumptions are

(10) $(\forall x)(x \in A \rightarrow x \in B)$,

and that (7) follows from (10) and the definition of subset.

We propose that this be dealt with in the following way. If the user wants to prove something in the form of (10), while making the assumption (8) he also states the goal (9). When the goal is legitimately achieved, the proof monitor generates formula (10) and associates it with the line (9). (At the present time a lot of additional information is being associated with each line, i.e., free variables, dependencies, etc.). Then if a reference to that line is made there will be no ambiguity, since the form (9) depended on (8) and is no longer available. (The working premise (8) was discarded -- see [8] -- and all lines depending on it are not available).

In more complicated proofs the user will have to state goals on different levels in the proof. Thus the monitor will deal with a hierarchy of embodied goals. It would be better if the goals were declared automatically, but at the present time we do not foresee any reasonable way of implementing the goal hierarchy handling without the user stating the goals. Top-level goals would not need to be explicitly mentioned.

An Informal Proof

We try to illustrate some of the above discussion with the informal proof of a simple theorem. The definition of image in the proof below is:

```
(∀R,A,y)(y ∈ R"A → (∃x)(x ∈ A ∧ <x,y> ∈ R)).

Theorem   C⊆D → f"C⊆f"D

Proof.
Assume
    (1)   y ∈ f"C
By the definition of image
    (2)   (∃x)(x ∈ C ∧ <x,y> ∈ f)
From (2), the hypothesis of the theorem, and
the definition of subset it follows that
    (3)   (∃x)(x ∈ D ∧ <x,y> ∈ f)
Whence by the definition of image
    (4)   y ∈ f"D
Q.E.D
```

The proof checker handles the steps as they are given here. Of course, accepting this proof would require stronger natural language processing capabilities than there currently are available in the program. The semantic analysis of the text should show that steps (2) and (4) are applications of the IMPLIES rule, while step (3) requires a call to the prover via VERIFY.

Future Directions

The problems facing future work on understanding informal proofs may be best illustrated by considering another comparative example. A proof of Theorem 37 on page 104 in [9] is presented below as it is accepted by QUIP. (To fully benefit from the comparison the reader should look at the proof in the book).

```
Derive: Finite (A)∧(∃f)f : A SURJ B → Finite (B)

Hypothesis
    (1) Finite (A) ∧ (∃f)f : A SURJ B
1C2 ES  [Line 1, Conjunct 2, Exist. Specific.]
    (2) f : A SURJ B
Axiom of Separation
    (3) (∀x)(x ∈ K ⟷ x ∈ P(A) ∧ Finite(f"x))
Th. 1.6, Th. 1.7, Th. Image  VERIFY
    (4) f"0 = 0
4, Th. 4.1.1  VERIFY
    (5) Finite(f"0)
3,5 Th. 1.48  VERIFY
    (6) 0 ∈ K
Assume
    (7) x ∈ A ∧ C⊆A ∧ C ∈ K
7, Th. Singleton, Df. Subset  VERIFY
    (8) {x} ⊆ A
7C3, Th. Concretion  IMPLIES
    (9) C⊆A ∧ Finite(f"C)
BOOLE UG
   (10) (∀A,B,C)(A⊆C ∧ B⊆C → A∪B ⊆ C)
8,9,10  VERIFY
   (11) C∪{x} ⊆ A
2,7, Th. 3.2.35, Df. Surjection, Df. Map  VERIFY
   (12) (∃y)f"({x}) = {y}
12, Th. 4.1.2  VERIFY
   (13) Finite(f"{x})
9,13, Th. 4.1.6  VERIFY
   (14) Finite(f"C ∪ f"{x})
14, Th. 3.2.29  RE  [Replace by Equality]
   (15) Finite(C ∪ {x})
3,11,15, Th. Power Set  VERIFY
   (16) C ∪ {x} ∈ K
7,16 CP UG  [Cond. Proof and Univ. Generaliz.]
   (17) (∀C,x)(x ∈ A ∧ C ∈ K → C∪{x} ∈ K)
1C1,6,17, Th. 4.1.11  IMPLIES
   (18) A ∈ K
18,4  IMPLIES
   (19) A ∈ P(A) ∧ Finite(f"A)
2, Df. Surjection, Df. Map, Th. 3.2.20  VERIFY
   (20)  f"A = B
19,20  VERIFY
   (21) Finite(B)
1,21  CP
   (22) Finite(A) ∧ (∃f)(f:A SURJ B) → Finite(B)
Q.E.D.
```

The proof in the book is given in a very compact form and requires a lot of thinking from the reader in order to follow it in detail. Many readers would probably agree with it after glancing through, but when asked for the justification of a certain step they might have difficulties in explaining it. Even a simple thing as defining the set K by abstraction really appeals to the axiom of separation to make sure that the set exists.

Yet in view of the applications we have in mind the proof checker of QUIP is more like the kind of system one needs. First, it is an important part of the task of a proof reader to make sure that all steps in a proof were properly justified. Second, in teaching axiomatic mathematics it is important to insure that the student understands the exact justification of a proof he might read in a book.

The knowledge necessary for understanding a proof as the one above will be distributed over the machinery for making large steps, the goal monitor, and the semantics of the mathematical language processor. Procedural knowledge about different ways of organizing proofs will be needed.

Most notably the QUIP proof in the last example seems to be organized in reverse order of the book proof. The latter first states that induction is needed to show f"A ε K, whence B is finite, since f"A = B, and then proceeds to develop the induction conditions. The QUIP proof works out the induction by Theorem 4.1.11 (Step 18) and then shows the finiteness of B. In fact, the proof includes several occurrences of the same phenomenon on a smaller scale.

Processing the wide variety of ways of expression seen in the mathematical literature is a formidable problem. It will require continuous work for time to come. But by putting some restraints on the form of presenting proofs one might soon be able to make good use of a computer system for the purposes outlined in the beginning of this paper.

Acknowledgements

Many of the presented ideas were formed in conversations with P. Suppes, L. Blaine, H. Graves, and R. Smith. Also, W. Bledsoe has made many helpful suggestions. The research has been supported by the U.S. National Science Foundation under Grant EC43997 to Stanford University.

References

1. Abrahams, P., "Machine Verification of Mathematical Proof", 1963, Doctoral Dissertation, MIT.
2. Goldberg, A., "A Generalized Instructional System for Teaching Elementary Mathematical Logic", Tech.Rep.No.179, 1971, IMSSS, Stanford University.
3. Marinov, V., "Maximal Clause Length Resolution", 1973, Doctoral Dissertation, University of Texas at Austin.
4. Marinov, V., "Replace Formula", 1974, IMSSS internal memo, Stanford University.
5. McCarthy, J., "Computer Programs for Checking Mathematical Proofs", 1961, Proc.Amer.Math. Soc. on Recursive Function Theory, New York.
6. Smith, R., Graves, H., Blaine, L. and Marinov, V. 1975. "Computer Assisted Axiomatic Mathematics: Informal Rigor." In: Computers in Education, 0. Lecarne and R. Lewis, eds. IFIP Second World Conference on Computer Education. North Holland, Amsterdam, pp.803-809.
7. Smith, R., "B00LE--A Decision Method", 1974, IMSSS internal memo, Stanford University.
8. Suppes, P., Introduction to Logic, 1957, Van Nostrand, Princeton, N.J.
9. Suppes, P., Axiomatic Set Theory, 1972, Dover, New York.

Appendix

Axioms, Definitions, and Theorems References in the Proofs.

Ax. Separation
$(\forall C)(\exists B)(\forall x)(x \in C \leftrightarrow x \in B \wedge FM(x))$

Ax. Cardinals
$(\forall A,B)(K(A) = K(B) \leftrightarrow A \approx B)$

Df. Subset
$(\forall A,B)(A \subseteq B \leftrightarrow (\forall x)(x \in A \rightarrow x \in B)$

Df. Map
$(\forall f,A,B)(f:A \rightarrow B \leftrightarrow Func(f) \wedge D(f)=A \wedge R(f) \subseteq B)$

Df. Surjection
$(\forall f,A,B)(f:A\ SURJ\ B \leftrightarrow f:A \rightarrow B \wedge R(f)=B)$

Df. Cardinal Number
$(\forall x)(Cardinal(x) \leftrightarrow (\exists A)(K(A) = x))$

Th. 1.6
$(\forall x)(x \notin 0)$

Th. 1.7
$(\forall A)((\forall x)(x \notin A) \leftrightarrow A = 0)$

Th. Concretion
$(\forall y)(y \in \{x:FM(x)\} \rightarrow FM(y))$

Th. Singleton
$(\forall x,y)(y \in \{x\} \leftrightarrow y = x)$

Th. Power Set
$(\forall A,B)(B \in P(A) \leftrightarrow B \subseteq A)$

Th. 1.48
$(\forall A)(0 \in P(A))$

Th. 3.2.20
$(\forall A,B,R)(R \backslash A \cap B = R \backslash A \cap R \backslash B)$

Th. Image
$(\forall A,B,y)(y \in A''B \leftrightarrow (\exists x)(<x,y> \in A \wedge x \in B))$

Th. 3.2.29
$(\forall R,A,B)(R''(A \cup B) = R''A \cup R''B)$

Th. 3.2.35
$(\forall f,x)(Func(f) \wedge x \in D(f) \rightarrow (\exists y)(f''\{x\} = \{y\}))$

Th. 3.8.9
$(\forall A,B)(\exists C,D)(A \approx C \wedge B \approx D \wedge C \cap D=0)$

Th. 4.1.1
$Finite(0)$

Th. 4.1.2
$(\forall x)\ Finite(x)$

Th. 4.1.6
$(\forall A,B)(Finite(A) \wedge Finite(B) \rightarrow Finite(A \cup B))$

Th. 4.1.11
$(\forall A,D)(IF\ Finite(A) \wedge 0 \in D \wedge (\forall B,x)(x \in A \wedge B \subseteq A \wedge B \in D \rightarrow B \cup \{x\} \in D)\ THEN\ A \in D)$