A PLATO PROGRAM FOR
INSTRUCTION AND DATA COLLECTION
IN MATHEMATICAL PROBLEM SOLVING

Prepared by
J. A. Easley, Jr., H. M. Gelder, W. M. Golden*

CSL REPORT R-185 JAN., 1964

COORDINATED SCIENCE LABORATORY
UNIVERSITY OF ILLINOIS
URBANA, ILLINOIS

DA-36-039-AMC-02208(E)
DA Project 3A-99-25-004
US NSF G23554

This research was made possible by support extended to the University of Illinois, Coordinated Science Laboratory jointly by the Department of the Army, Department of the Navy (Office of Naval Research), and the Department of the Air Force (Office of Scientific Research), under Department of Army Contract DA-36-039-AMC-02208(E) and by support extended to the University of Illinois, UICSM Mathematics Project, by the National Science Foundation under Contract No. US NSF G23554.

* The authors are staff members of the UICSM Mathematics Project, 1210 W. Springfield, Urbana, Illinois.
1. Introduction

One of the most challenging tasks in automated instruction in mathematics is that of developing a system for teaching proof. The challenge lies primarily in the following three problems:

First, there are, in general, many different sequences of statements which constitute proofs of a given theorem. Furthermore, there is no generating rule by which one could set down all possible proofs ahead of time, and, even if all possibilities were known, it would be difficult to design an efficient checking procedure based on stored proofs. In an automated teaching system, then, if the student is to be free to write any proof he can, how is it to be decided whether what he has written is a proof or not?

Second, a sequence of statements which seems to the student to connect what is given and the theorem to be proved, and which also serves the psychological function of helping the student to find the conclusion, still may not be a proof. According to a strict logistic point of view, the steps of a proof are supposed to form a logical chain, linking the premises and the conclusion. Even according to customary practice in writing proofs, which allows gaps in the logic and imprecisely formed statements, the arguments must convince knowledgeable persons other than the writer. So, writing down a series of steps which terminates in the desired statement is not a sufficient achievement to be counted as a proof. Consequently, self-judgement by comparison of the terminal expression with the goal expression is impractical.

1 The authors are especially indebted to Donald L. Bitzer for coding the computer program and to Patricia Cutler who, until her untimely death, helped us in so many ways, especially by preparing program tapes and in operating the computer during system checks. We are also grateful to Joel Selig for assistance in preparing parameter tapes.
Third, if the student is to be informed of his errors when he makes them, checking needs to be performed at every step of his proof. Yet, following any given sequence of steps in a proof, there may be many legitimate next steps, some of which may be useless and hence not found in any stored proof. So, the checking difficulties noted for entire proofs are compounded, when it is attempted to perform checks at each step.

These same difficulties arise in connection with other mathematical problems such as simplification of expressions and solution of equations. They arise, in fact, whenever the student’s work is expected to justify his answer completely. There are often many correct ways of arriving at the solution of a given problem, and also some incorrect ways of reaching it. The claim that writing instructional programs is easy in mathematics because there are unique answers to problems, appears to lose its validity as one looks closely at the task of guiding students while solving challenging problems. There is one consolation, however, a correct method does not yield a wrong answer.

One step toward the solution of these programming problems has been taken in the development of a special teaching logic for the PLATO II computer-based teaching system. This teaching logic, called PROOF, takes advantage of the flexibility and student-controlled sequencing made available in the laboratory or inquiry mode of programming PLATO. There are two limitations on the general applicability of PROOF to mathematical problem solving. The first limitation can be removed by a relatively simple modification of the computer program, and the second limitation, which may really be a blessing in disguise, would require a much more elaborate computer program to remove.

The first limitation of PROOF is that the student can only perform proofs or derivations which form a linear chain — that is, each step follows from the preceding step by virtue of some axiom or theorem. But a modified program is under development in which branching proofs or derivations can be constructed — that is, a given step can be made to

---

follow from any one, or any two, previous steps. The demands of various applications on this more general teaching logic are considered in the fifth section of this report.

The second limitation lies in the fact that, in PROOF, the particular instance of the axiom or theorem which justifies each step must be specified in some detail by the student. This means in practice, that the student is forced, because of the procedures needed to operate the machine, to construct a complete proof. That this could be considered a limitation may be surprising in view of the fact that mathematics is often advertised as the subject in which all arguments are supported by the principles of logic. In actual practice, however, proofs and problem solutions are usually written in an abbreviated form in which most of the indicated inferences do not follow by direct application of the principles of symbolic logic. Such inferences would themselves require derivations of several steps each. However, constructing proofs in a complete form can be a very interesting and worthwhile endeavor — as has been demonstrated by the UICSM mathematics project.³

To be sure, the student must eventually learn to write proofs in a less complete form. Most of the proofs of synthetic geometry, for example, would be much too long for practicality if they had to be written out completely. But, having the experience of actually constructing complete proofs should enable the student to conceive realistically of the requirements that would have to be met if he were to construct a complete proof along the lines of a conventional geometry proof. This conception could then serve as a means of checking the acceptability of the shortcuts which he takes as a matter of practical necessity. Thus, before students reach the stage at which they are called upon to write proofs in abbreviated form, it seems especially desirable to introduce them to the study of complete proofs.

We do not know if it would be possible to prepare a teaching logic for proof-writing in which the validity of all shortcuts typically taken in conventional proofs could be evaluated by the computer. Because of the

numerous pedagogical advantages to be gained by requiring rigorous proofs, we have no immediate motivation for developing such a program. However, it is interesting to note that Newell, Shaw, and Simon have developed a heuristic program for a computer so that it constructs proofs of a half-dozen steps or so in a few minutes. Recent work by Wang has demonstrated the feasibility of generating such proofs in a matter of seconds. This is roughly the same number of steps which a student might combine in one line of a typically written solution to an equation. By incorporating the methods developed by these investigators, it is possible in principle to develop a revised teaching logic for mathematical problem solving which would judge the validity of some conventional shortcuts. One must consider the possibility that shortcuts which are conventional in proofs in geometry and perhaps in some other fields, may exceed the practical limitations of a proof-constructing program of the sort developed by Newell, Shaw, and Simon. Even so, there may be other benefits from incorporating this sort of program, such as the capability of evaluating the strategy or elegance of a proof as well as its logical validity.


2. Educational Uses of the PLATO-PROOF System.

The present PROOF program has been used to give supplementary lessons in the mechanics of proof to junior high school students and to high school teachers who are studying UICSM texts. It is hypothesized, both in the case of students learning proof for the first time and in the case of teachers who have known only abbreviated proof-writing, that experience in constructing proofs with the PLATO-PROOF system should make clear to them a mechanical procedure by which they can certify that a line of proof follows from its premises. This procedure is the student's mental analog of the actual procedure which would be used by the computer to check the validity of that same line. Furthermore, experience with this system should structure a student's approach to the task of mathematical deduction so that he views the creative aspects of this task as deciding on the answers to a repeated cycle of questions — what transformation rule, which variable, what substitution? However, we have not been able to collect sufficient data as yet to draw firm conclusions about the teachability of proof by machine.

Proofs of theorems of algebra appear to be very useful for both of these above purposes. Although it is not customary to introduce proofs in elementary algebra, such proofs have the advantage of being both simple and direct even when written out in a complete form. However, algebra is certainly not the only possible subject for an introduction to rigorous proof. (Other applications of the program will be considered briefly in Section 5 of this report.) In order to illustrate the conception of the mechanics of proof, which experience with the PLATO-PROOF system could give a student, a brief outline of the PROOF operations is presented.

In the PROOF teaching logic there are five major operations which the system places at the student's command:

(1) The student can select which of a given set of problems he wishes to work first. At times, selecting an efficient order in which to solve a set of problems may be an important part of his general problem-solving activity.

(2) The student can form an instance of any general principle stored in the computer, replacing each variable with constants, other
variables, or expressions of his choice. For certain purposes, the problem itself may need to be recast by substitution — in other cases the most difficult part of a problem may be the recognition that an unusual instance of a given generalization will form a needed link in solving a problem.

(3) The student can apply an instance of a generalization (or a hypothesis stated in the problem) to transform a statement or expression which he has already derived.

(4) The student can copy untransformed portions of the statement or expression last derived. This permits principles and hypotheses to be applied in the appropriate places without going through a formal application of the propositional calculus to entire statements.

(5) The student can ask the system to judge whether solution to the problem is acceptable. In a completed proof, the final line derived must match exactly the line representing his goal.

The teaching logic is such that the production of a matching line guarantees that the student's entire derivation constitutes a complete mathematical proof. The system can then inform the student of his success and record the theorem proved as one that he can call upon in the future. We turn now to a somewhat more detailed discussion of the computer program which performs these operations.
3. Structure of the Computer Program for PROOF

Any system which, like algebra, involves short strings of symbols and fixed rules for transforming one string of symbols to another can be handled by this program or some minor variation of it. Symbolic logic is such a system. So is Boolean algebra. Even within the field of algebra, one is not restricted by the program to exercises in algebraic proofs. Indeed, in equation solving and simplification of algebraic expressions, transformation principles are used in the same way as in proofs. Thus, while we now describe the essential features of the program in terms of its use in algebraic proofs (primarily because we are most experienced in thinking in those terms), in a later section, we will consider the use of the system for other types of problems in algebra and in other branches of mathematics.

The PROOF teaching logic is a program coded in ILLIAC language and run with the PLATO II equipment by the CSL Control Data 1604 computer. The PROOF program is accompanied by the PLATO LAB program and by SIMILE, the ILLIAC simulation routine. It is also accompanied by statements of the exercises, the generalizations which are to be available to the student solving the problems, and other coded information particular to the lesson desired.

In order to accomplish the major operations described above, the program reserves 50 words of memory for each of two students. Nineteen of these 50 words are reserved for various constants necessary for the operation of the many sub-routines which comprise the program. The remaining 31 words are divided into four memory banks labeled P, SP, SSP, and SPS respectively. When the program is in operation, these four banks contain coded expressions pertinent to the exercise being worked. Alphabetic symbols, punctuation signs, mathematical or logical operation signs, etc., are each arbitrarily assigned a six-binary-digit code. Within the computer, the strings of symbols corresponding to an exercise or one of the steps in its solution are represented by corresponding strings of these six-digit codes.

When the student selects a theorem, it is stored in the P memory bank precisely as given. Later, the P bank contains the exercise stated in terms of variables freely chosen by the student. Thus, Figure 1-A

(text continues on p. 11)
The student has selected a theorem to be proven. P initially contains the theorem as given.

The student has elected to replace 'x' by 'a' and 'y' by 'b' in constructing his proof.
The student has selected as his transformation rule the principle of subtraction [abbreviated ps on screen]: \( \forall x \forall y (x + (-y)) = (x - y) \). A coded statement of the rule is automatically entered into SPS from the stored list of transformation rules.

The student replaced 'x' and 'y' in the transformation rule [as stored at time \( t_3 \)] by 'b' and '1' respectively, indicating how he intends to use the rule. Note that the left-hand expression in SPS matches characters two through nine in SP.
The matching procedure at time $t_4$ was successful. The computer plots '$-1$' on the screen and stores it in SSP.

The student had started with a copy of the left member of the identity, as stored in P, and his last line is now a copy of the right member. The computer checks the last line of the proof against P before showing the slide containing the conclusion.
illustrates the initial contents of \( P \) copied at time \( t_1 \) from the main computer memory where the theorem had been stored by the author of the lesson. This figure and subsequent figures also show the appearance of the screen which is indicated by a frame. Figure 1-B shows the expressions stored in \( P \) at time \( t_2 \) when the student completed his selection of the variables he will use in proving the theorem, and it also shows the first line of proof which was then plotted on the student's screen. Note that from time \( t_2 \) until the completion of the proof, \( P \) contains the expression toward which the student is working. (Compare contents of \( P \) in Figures 1-B, 1-C, and 1-E with the next to the last line of the proof shown in Figure 1-F.)

The SP memory bank contains that line of the student's proof which was most recently derived. Thus, for example, in Figures 1-C, 1-D, and 1-E the content of SP is line four of the proof, the last completed line. The fifth line of the proof will be the result of applying one of the transformation rules to part of line four, the remainder of that line being copied.

The SSP bank contains that part of line five, the line currently being developed, which is already on the screen. Thus, in Figure 1-C, SSP contains three characters copied by the student from the line above and, in Figure 1-E, SSP contains three additional characters resulting from application of the transformation rule.

The SPS memory bank is intimately connected with the functioning of those sub-routines which accomplish the application of transformation rules. In it, the instance of the principle which is to be used in transforming a line of proof is found. Figure 1-C shows the contents of the SPS at time \( t_3 \) just after the student has indicated his intention to use the principle of subtraction to transform a portion of line four. Figure 1-D shows the contents of the SPS at time \( t_4 \), when the student has indicated the way in which he intends to use this transformation rule, i.e., he had indicated the expressions (in this case single characters) to be substituted for each of the variables in the transformation rule as given. At this time, \( (t_4) \), the program takes over control from the student. The left-hand expression in SPS is checked character for character against the appropriate portion of the stored line in SP. If the expressions match, as in the case illustrated in Figure 1-D, then the right-hand portion of the expression in SPS is plotted at the appropriate place on the screen (refer to the last line of

(text continues on p. 14)
Student has indicated that he will use the uniqueness principle for multiplication [upm: $\forall x \forall y \forall z \ x = y \implies (xz) = (yz)$] to generate the second line of the proof.

The student replaced the 'x's', 'y's' and 'z's' in the transformation rule as stored in SPS by '(b + (-1))', '(b - 1)', and 'a', respectively. Note that the left hand expression in SPS matches contents of SP.
Line two is plotted automatically after checking procedures are completed. Note that the inference completed here is an application of the law of detachment (or modus ponens), \((p \land (p \rightarrow q)) \rightarrow q\). Since the computer carries out the same operations whenever it transforms a line or part of a line, all inferences within the body of proofs can be regarded as instances of a modus ponens-like rule.

The remaining steps in the proof are shown in Figure 2-D.
the partial proof in Figure 1-D) and stored in SSP. The computer is able to determine that a part of the right-hand expression in SPS is already on the screen, and does not replot that portion of the expression. If the left-hand expression in SPS is found to differ from the characters in SP, the keyset "beeps" to indicate to the student that he has made an error and control is returned to the student.

Figures 2-A, 2-B, and 2-C show the functioning of the SPS in the application of a transformation rule which applies to an entire equation. The reader should note that this transformation requires the substitution of several characters for two of the variables in the transformation rule as given. In making these substitutions, the constants which indicate the line length and the lengths of the two sides of SPS must be adjusted accordingly.

With the preceding discussion available as background, it will be possible to illustrate briefly the remaining features of the PROOF program.

Copy Button:

Generally, in constructing a line of a proof, only parts of the preceding line are to be transformed. A button is provided on the keyset which, when pushed, automatically plots the next character out of SP, and stores it in SSP. The first three characters of line five of the partial proof shown in Figure 1-C were copied in this manner.

Hypothesis Button:

In proving a conditional theorem such as that shown in Figure 3, one extra transformation rule which involves making use of the hypothesis is available in addition to those given in the stored list of transformation rules. The student initiates this transformation by pushing the hypothesis button on the keyset. A coded statement of the hypothesis is transferred to SPS and compared with the appropriate expression in SP. If this matching procedure is successful, the transformed expression is plotted on the screen. If the expressions do not match, or if for some other reason it would be improper to apply the transformation, the keyset "beeps".
Conclusion Button:

When the student believes he has completed a proof, he pushes the conclusion button. The computer checks for a completed proof by comparing the student's last line (which, as always, is to be found in SP) with part or all of the stored expression in P. The part of P to be used in the comparison depends upon the type of theorem and the strategy of the proof. If the theorem is a conditional generalization (Figure 3), the last line of the proof should match the right side of P. If the theorem is an identity and the student began with the left member, his last line should match the right half of P (Figure 1-F). Given the same identity to prove, if the student starts with the right member, his last line should match the left side of P. Finally, if the student begins his proof of the identity with an instance of some generalization (as in Figure 2-D), the last line of the proof should match the entire expression in P. The computer selects the correct procedure in each case, and shows the conclusion slide to the student if the proof was correct; otherwise the keyset "beeps" and the student continues his proof.

Special Requests:

(a) Continue: If a line of the proof is too long to fit on the screen, pushing the continue button allows the student to move down to the next line of the screen and to indent six spaces. If the computer is automatically plotting an expression which would extend beyond the right-hand margin of the screen, it too continues on the next line with an indentation.

(b) Erase: Pushing the erase button completely clears the PLATO screen preserving only the student's choice of exercise.

(c) Scratch: Pushing the scratch button produces a dashed line through the current line of proof. This permits the student to change his mind on one step without having to start from the beginning of his proof as he must when he pushes the erase button. The computer will scratch out two screen lines if the continue button has been pushed during the development of the current line.
(d) Symmetry of Equality for Hypothesis: In Figure 3, pushing the hypothesis button resulted in replacing 'b' in line one by 'a' in the corresponding position on line two. Had the student desired to replace 'a' by 'b', it would have been necessary for him to first push the symmetry of equality for hypothesis button so as to restore the hypothesis in its symmetric form, i.e., 'a = b' instead of 'b = a'.
Note: In this illustration we have eliminated the replacement of each of the variables given in the exercise and the grouping symbols to better show the functions of the hypothesis button.

Theorem: \( b = a \implies ac - bc = 0 \)

\[
\begin{align*}
ac - bc &= ac - bc \\
ac - bc &= ac - ac \\
ac - bc &= 0
\end{align*}
\]

So, \( b = a \implies ac - bc = 0 \)

This very elegant proof requires that the student use the hypothesis to replace the 'b' in the right-hand side of line 1 by an 'a'. This can be accomplished by pushing the hypothesis button at the appropriate time. The contents of the four memory banks at that time are shown below.

<table>
<thead>
<tr>
<th>P</th>
<th>SP</th>
<th>SSP</th>
<th>SPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = b</td>
<td>ac - bc = ac - bc</td>
<td>ac - bc = ac - a</td>
<td></td>
</tr>
<tr>
<td>ac - bc = 0</td>
<td>(line 1)</td>
<td>(part of line 2)</td>
<td></td>
</tr>
</tbody>
</table>

The 'b' in the left-hand part of SPS matches the 'b' in the SP so an 'a', the expression in the right-hand part of SPS, is plotted.
4. The Proof Keyset

The student employs a small ten-button keyset to communicate to the system the various decisions that he makes in constructing a proof of a theorem. By pressing the keyset buttons in certain, proper, well-formed sequences the student can accomplish:

a. The selection of a particular theorem from the set offered in the lesson, his choice of the offered strategies for attacking the proof, and the formation either of an instance of a member of the theorem or of an assumed principle with which he wishes to begin.

b. Each of the steps of a proof by calling upon a transformation principle and making proper substitutions for the variables in it, which, by a "modus ponens" logic, will produce the desired inference.

c. Such special operations, discussed in the previous section, as copy particular characters, scratch out a line, erase the screen, announce the completion of work in a particular line, assert that a proof has been formed, and so forth.

As will be noted from an examination of Figure 4, which gives a picture of the keyset, each of the buttons other than START and OFF serve several distinct purposes. Generally, to each button has been assigned a particular numeral (far right column of the keyboard), a variable name in the variables column, a character in each of the two halves of the substitutions column, and a function in the control column. Four buttons have an additional assignment in the special request column at the far left.

At the base of each of the four full columns of the keyboard is a small light bulb. If the next decision the student should make is that of selecting an exercise theorem, a proof strategy, or a transformation principle — which are identified by numerals — the light would be on under the numerals column. Were the next step one of designating which of the variables of the theorem or of a transformation principle is to be substituted for, the bulb at the base of the variables column would be lighted. The characters which will be used to form an expression to be substituted for a particular variable are to be chosen in proper sequence from the left and right halves of the third column, and the bulb at the base of this column will be on while the choice of these characters is being made.
Figure 4. The PROOF Keyset
To shift from the right to the left half of this column, the SET 2 button is pressed. To indicate the completion of the expression desired, the FINISH SUBSTITUTION button is pressed. Finally, if the bulb at the base of the control column is lighted, the system will perform that function which is indicated by the next button pressed.

This system of lights helps to inform the student concerning what meaning the device will give to the next button pressed in a sequence. The light is programed to shift automatically from one column to another according to what the teaching logic decrees is to be the next kind of decision that the student should make. The lights, therefore, provide him with some cues for the sequence of steps in his task.

The keyset will emit a short tone, or "beep", if the student should press an "illegal" button or sequence of buttons at any point in the lesson, but it remains to the student to determine what error he has committed. What is to be "illegal" at a particular point in constructing a proof is sometimes a subtle problem which the designer of the teaching logic must face and will not be discussed here. However, certain examples were given in the previous sections and others will occur to the reader. For example, the designation of a variable irrelevant to the theorem or transformation principle, an attempt to substitute an incomplete or improper expression for a variable, or a request to copy a non-existing character are mistakes which would result in a prompt "beep".

To begin a lesson the student presses the START button and obtains, on the screen, the set of exercise theorems. He makes his selection of one of this set by pressing the button opposite the numeral of that theorem. Next displayed on the screen is the theorem of his choice and a set of instructions on how to select one of the possible proof strategies. According to this choice, he may be directed to form an instance of one or the other of the members of the theorem, or of a principle from which the student believes he can deduce the theorem. By pressing the numeral button which corresponds to his choice of strategy, the screen is changed to a "blackboard" with the theorem to be proved written at the top. The student then carries out the first step according to the instructions that were given and proceeds to the presentation of his proof.

An illustration of the procedures followed in selecting and making use of a principle to transform one line of a proof to the next should be sufficient to give the reader a reasonably complete understanding of how the
sequence of buttons to effect the transformation is formed. We will refer to the example of a complete proof which is displayed in Figure 1-F. The second line of this proof is the expression:

\[ (((-1)a) + (ba)) = \]

This is transformed to the expression on the third line by means of the distributive principle of multiplication over addition (dpma) which is among the generalizations stored for use with this lesson.

\[
dpma \quad \forall x \forall y \forall z \quad ((xz) + (yz)) = ((x + y)z)
\]

Comparison of the expression to be transformed with the left member of the transformation principle indicates that the variable ‘x’ should be replaced by ‘(-1)’, the variable ‘y’ by ‘b’, and ‘z’ by ‘a’. The following tableau demonstrates the sequence of button presses which will accomplish this.

<table>
<thead>
<tr>
<th>Step</th>
<th>Light in Column</th>
<th>Button</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Numerals</td>
<td>2, then 5</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Variables</td>
<td>x</td>
<td>Code name of principle</td>
</tr>
<tr>
<td>4</td>
<td>Substitution</td>
<td>(</td>
<td>In set 1</td>
</tr>
<tr>
<td>5</td>
<td>Substitution</td>
<td>-</td>
<td>In set 1</td>
</tr>
<tr>
<td>6</td>
<td>Substitution</td>
<td>Set 2</td>
<td>Shifts to left half.</td>
</tr>
<tr>
<td>7</td>
<td>Substitution</td>
<td>1</td>
<td>Automatically shifts back to right half.</td>
</tr>
<tr>
<td>8</td>
<td>Substitution</td>
<td>)</td>
<td>In set 1</td>
</tr>
<tr>
<td>9</td>
<td>Substitution</td>
<td>Finish</td>
<td>Completes replacement for ‘x’.</td>
</tr>
<tr>
<td>10</td>
<td>Variables</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Substitution</td>
<td>Set 2</td>
<td>Shifts to left half.</td>
</tr>
<tr>
<td>12</td>
<td>Substitution</td>
<td>b</td>
<td>Automatically shifts back to right half.</td>
</tr>
<tr>
<td>13</td>
<td>Substitution</td>
<td>Finish</td>
<td>Completes replacement for ‘y’.</td>
</tr>
<tr>
<td>14</td>
<td>Variables</td>
<td>z</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Substitution</td>
<td>Set 2</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Substitution</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Substitution</td>
<td>Finish</td>
<td>Completes replacement for ‘z’.</td>
</tr>
</tbody>
</table>
The student having designated replacements for all of the variables of the principle, the computer makes the check described in Section 3, which succeeds, and then it plots this expression on the third line of proof:

\[ (((-1) + b)a) \]

To complete the line the student presses the COPY button to plot an '=' sign and then the FINISH LINE button to shift to the next line on the "blackboard". The student proceeds in the same fashion until he has generated the line that represents his goal. He then presses the button marked FORM CONCLUSION and a slide change brings on the concluding statement of the theorem proved. Then, by depressing the START button he is presented with the list of exercises, and he is ready to choose his next exercise.
5. Other Applications

In order to illustrate the diversity of possible applications of PROOF and the generality of its basic routines, we now give attention to possible adaptations of the PROOF program to mathematics problems other than those of proving elementary theorems in algebra. It has already been pointed out that adaptations for handling simplification problems and the solution of equations would require no basic changes in the PROOF program. In both cases, the work normally proceeds in a linear chain of transformations, each transformation applying a theorem of algebra or fact of arithmetic. The stored list of useful theorems may be somewhat larger than one ordinarily employs in proofs and the facts of arithmetic required in a given set of problems could be quite large. However, judicious lesson planning may be able to hold the lists within reason. In the case of simplification problems, it is sometimes necessary to allow any one of two or three alternative expressions as a satisfactory simplification.

Finding the solution sets of simple inequalities should be no more difficult than solving equations, provided the appropriate transformation rules are made available. If the student were allowed to enter the expression to be simplified or the equation to be solved, worded problems could be presented instead of algebraic expressions. In the case of the worded problems, however, it would be desirable to provide a check of the expression formed by the student.

In order to extend the usefulness of the basic routines, the general mathematical problem-solving program should be a teaching logic in which it is easy to vary the extent to which decisions are left to the student. The optimum degree of student control will depend a great deal on the subject matter, the level of experience of the student and the purpose for which the program is being used. By experimenting with the degree of automation, one may be able to determine the most suitable degree for studying problem solving and also to discover effective schedules for changing the automatic features in the course of a teaching program.
There are at least two aspects of student control which, although related, need to be considered separately. They are the completeness with which the student must specify his desires at each step, and the flexibility of choice he has in what he can do next. Each of these aspects in turn has two sub-aspects — the specificity and flexibility can be modified either in the completeness of proof or in the variability of the order in which things must be done. The most plausible approach to this objective is to design the program assuming maximum student control of all aspects but providing for the automation of various decisions at the lesson writer's discretion. For purposes of introductory lessons, with simple problems, it should be possible to simplify the student's task by automating certain decisions. On the other hand, automation could also enter into sophisticated versions to relieve the burden of specificity required.

The present PROOF program is already highly automated. The most desirable next goal would be the production of a teaching logic which increases student control as much as is possible, but permits easy resetting to the present level of automation. It should also add additional features which give generality without interfering with the operation in simple cases. The detailed planning of such a teaching logic is essentially complete, and programming it has already begun.

After gaining experience with this new teaching logic, we should be able to judge the advisability of developing another new program which would increase student control as much as could conceivably be desired — provided again that satisfactory means can be found for returning the program to a level of automation like that in PROOF.

To illustrate some of the desirable range of student control and its uses, consider the case of a student solving the equation $3x^2 + 18x = 21$. His work might look like the following after he had finished:
\[ 3x^2 + 18x = 21 \]
\[ x^2 + 6x = 7 \]
\[ x^2 + 6x - 7 = 0 \]
\[ (x + 7)(x - 1) = 0 \]
\[ x = -7 \text{ or } x = 1 \]

The extreme of automation which we would consider developing would allow the student to indicate commands like those shown in the right-hand column above. After each command, the system would check that the command is legitimate and then print out the resulting line of the derivation. On the other hand, for some purposes, we may wish to provide a much higher degree of student control. It might very well be that some students — and perhaps all students at some stage — would learn better or reveal their thought processes better if they actually type out each line in whatever order they chose, identifying the transformation rule they are using for each, and, whenever they feel ready, ask the system to check if their work to that point is valid.

These two different ways of operating the system, and many others besides, should be readily available to the experimenter or instructional program writer. It must be recognized that solving problems on PLATO cannot completely resemble free pencil and paper work. In general, any means of collecting process data must interfere with the process to some extent. However, when the degree of freedom can be varied systematically so that different aspects of problem solutions come under control, it should become possible to ascertain the effect of data gathering on the thought processes.

A subject in which training in the mechanics of constructing complete proofs fits very naturally is symbolic logic. Some simple, linear proofs in logic can be written with the present PROOF program by changing some of the characters and by introducing logical rules in place of the axioms and theorems of algebra. One such proof is shown in Figure 5.
\[(q \lor \sim p) \land p \leftrightarrow q\]

\[
(q \lor \sim p) \land p
\]
\[
(\sim q \rightarrow \sim p) \land p
\]
\[
(\sim \sim p \rightarrow \sim \sim q) \land p
\]
\[
(p \rightarrow q) \land p
\]
\[
p \land (p \rightarrow q)
\]
\[
q
\]

RULES USED

\textit{ledi}  
Equivalence of disjunction and implication: \( p \lor q \leftrightarrow (\sim p \rightarrow q) \)

\textit{tcp}  
Law of contraposition: \( (p \rightarrow q) \leftrightarrow (\sim q \rightarrow \sim p) \)

\textit{ldn}  
Law of double negation: \( \sim \sim p \leftrightarrow p \)

\textit{clfc}  
Commutative law of conjunction: \( p \land q \leftrightarrow q \land p \)

\textit{ld}  
Law of detachment (modus ponens): \( (p \land (p \rightarrow q)) \rightarrow q \)

Figure 5. A linear proof in propositional logic.
However, branching proofs and derivations, in which several linear chains of inferences are joined together in a tree-like arrangement, are much more frequently encountered in symbolic logic than in elementary algebra. So, extensive use of problems in logic would require a teaching logic in which the student could structure the proof by identifying the particular line or lines he is calling on rather than having to follow a forced structure for his proof.

For examples of branching proofs in a format suitable for use with a more general mathematical problem-solving program, see Figures 6-A and 6-B. In branching proofs, it is also necessary to join two or more lines of proof to form a third. This procedure is illustrated in these figures wherever two line codes at the right of each proof are separated by a conjunction sign (∧) instead of a comma.

The two proofs shown in Figures 6-A and 6-B illustrate how a simple modification of the program can reduce the degree of specificity required of the student. In Figure 6-A, only rules of logic may be used to transform statements. In Figure 6-B, however, the student is also allowed to use premises or existing lines of proof as transformation rules. With this proviso, the two longest lines of the proof in Figure 6-A (line 12 and line 15) can be skipped. In the shorter proof, shown in Figure 6-B, instead of calling on the modus ponens rule, r2, the student derives line 12 by calling on the conditional statement 01 together with line 11. Line 14 of this proof is derived by calling on the conditional statement 13 together with premise 03. Thus, in the shorter proof, the built-in modus ponens procedure is used instead of calling on that rule explicitly. In principle, other rules can be built into the system, making other types of shortcuts possible, but once they have been built in their use can be easily restricted as desired for particular educational and research purposes.

Once we notice that problems in the various branches of logic can be handled by the PROOF program, it becomes apparent that applications of logic can be handled with only trivial changes. For example, switching circuits can be represented by Boolean algebra. Simplification of complex switching circuits is usually best accomplished by simplification of the corresponding Boolean expression. Elementary cases of
<table>
<thead>
<tr>
<th>Premises</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>(p \lor q) \rightarrow (r \lor s)</td>
</tr>
<tr>
<td>02</td>
<td>(q \lor r) \rightarrow t</td>
</tr>
<tr>
<td>03</td>
<td>\sim t</td>
</tr>
<tr>
<td>04</td>
<td>p</td>
</tr>
</tbody>
</table>

**PROOF**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>p \lor q</td>
<td>04, r1</td>
</tr>
<tr>
<td>12</td>
<td>(p \lor q) \land ((p \lor q) \rightarrow (r \lor s))</td>
<td>11 \land 01</td>
</tr>
<tr>
<td>13</td>
<td>r \lor s</td>
<td>12, r2</td>
</tr>
<tr>
<td>14</td>
<td>\sim t \rightarrow \sim (q \lor r)</td>
<td>02, r3</td>
</tr>
<tr>
<td>15</td>
<td>\sim t \land (\sim t \rightarrow \sim (q \lor r))</td>
<td>03 \land 14</td>
</tr>
<tr>
<td>16</td>
<td>\sim (q \lor r)</td>
<td>15, r2</td>
</tr>
<tr>
<td>17</td>
<td>\sim q \land \sim r</td>
<td>16, r4</td>
</tr>
<tr>
<td>18</td>
<td>\sim r</td>
<td>17, r5</td>
</tr>
<tr>
<td>19</td>
<td>(r \lor s) \land \sim r</td>
<td>13 \land 18</td>
</tr>
<tr>
<td>20</td>
<td>s</td>
<td>19, r6</td>
</tr>
</tbody>
</table>

**RULES USED**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>p \rightarrow (p \lor q) Rule of addition</td>
</tr>
<tr>
<td>r2</td>
<td>(p \rightarrow (p \rightarrow q)) \rightarrow q Law of detachment (modus ponens)</td>
</tr>
<tr>
<td>r3</td>
<td>(p \rightarrow q) \leftrightarrow (\sim q \rightarrow \sim p) Law of contraposition</td>
</tr>
<tr>
<td>r4</td>
<td>\sim (p \lor q) \leftrightarrow (\sim p \land \sim q) De Morgan’s Law I</td>
</tr>
<tr>
<td>r5</td>
<td>(p \land q) \rightarrow q Detachment for conjunction</td>
</tr>
<tr>
<td>r6</td>
<td>((p \lor q) \land \sim p) \rightarrow q Theorem 4</td>
</tr>
</tbody>
</table>

Figure 6-A. A branching proof in propositional logic using modus ponens explicitly.
<table>
<thead>
<tr>
<th>Premises</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>01 ((p \lor q) \rightarrow (r \lor s))</td>
<td>s</td>
</tr>
<tr>
<td>02 ((q \lor r) \rightarrow t)</td>
<td></td>
</tr>
<tr>
<td>03 (\sim t)</td>
<td></td>
</tr>
<tr>
<td>04 (p)</td>
<td></td>
</tr>
</tbody>
</table>

**PROOF**

<table>
<thead>
<tr>
<th>Line</th>
<th>Statement</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>(p \lor q)</td>
<td>04, r1</td>
</tr>
<tr>
<td>12</td>
<td>(r \lor s)</td>
<td>11, 01</td>
</tr>
<tr>
<td>13</td>
<td>(\sim t \rightarrow \sim (q \lor r))</td>
<td>02, r3</td>
</tr>
<tr>
<td>14</td>
<td>(\sim (q \lor r))</td>
<td>03, 13</td>
</tr>
<tr>
<td>15</td>
<td>(\sim q \land \sim r)</td>
<td>14, r4</td>
</tr>
<tr>
<td>16</td>
<td>(\sim r)</td>
<td>15, r5</td>
</tr>
<tr>
<td>17</td>
<td>((r \lor s) \land \sim r)</td>
<td>12 \land 16</td>
</tr>
<tr>
<td>18</td>
<td>(s)</td>
<td>17, r6</td>
</tr>
</tbody>
</table>

**RULES USED**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>(p \rightarrow (p \lor q))</td>
</tr>
<tr>
<td>r3</td>
<td>((p \rightarrow q) \leftrightarrow (\sim q \rightarrow \sim p))</td>
</tr>
<tr>
<td>r4</td>
<td>(\sim (p \lor q) \leftrightarrow (\sim p \land \sim q))</td>
</tr>
<tr>
<td>r5</td>
<td>((p \land q) \rightarrow q)</td>
</tr>
<tr>
<td>r6</td>
<td>(((p \lor q) \land \sim p) \rightarrow q)</td>
</tr>
</tbody>
</table>

*Figure 6-B. A branching proof in propositional logic using the built-in modus ponens logic as a shortcut.*
simplification in Boolean algebra can be performed with the present PROOF program as illustrated in Figure 7. A useful new feature for such an application would be a means whereby the student could compose an expression representing the diagram of the circuit to be simplified. With this feature, a means of judging the correctness of the expression composed (and of judging the completeness of the simplification) would need to be included. Possibly, a list of alternate expressions against which comparisons are made will suffice for these judge routines. More complex applications of logic will require the branching capabilities discussed above.

One customarily thinks of geometry in connection with proof, yet writing complete proofs is a far more difficult task in geometry than in most other branches of mathematics. Additional features already proposed for the new program, such as those permitting branching proofs and student formulation of expressions representing diagrams, would be useful in geometry. Some additional logical rules will need to be incorporated into the program instead of being treated as transformation rules. For example, it is frequently necessary in geometric proofs to form conditional statements from previous lines. It would be convenient to be able to do this by indicating a line number, an implication symbol, another line number, and, if the result is a valid inference, plot the statement so formed. It may even be desirable to automate such logical principles as the one identified in Figure 8 as TL 20, to avoid having to form lengthy substitutions on the keyboard.

A new problem is found in geometric proofs because of the convenient custom of using several different names for the same parts of a geometric figure. In the sub-routine which checks the contents of the SPS memory bank (see Section 3) against the SP bank, ∠BAC needs to be recognized as the same as ∠CAB, segment BC as the same as segment CB, and, with figures, triangle CDF as the same as triangle FCD, etc. One might wish to eliminate the complex programming required, if the computer is to accept alternate names, by requiring that the student name every segment and figure in alphabetical order (or in cyclical order), and that the first and last letters of angle names be alphabetized, but this requirement would lead to other problems as described below.
\[(a \lor (b \land c)) \lor ((\sim a \lor c) \land b) \leftrightarrow a \lor b\]

\[
egin{align*}
(a \lor (b \land c)) \lor ((\sim a \lor c) \land b) & \leftrightarrow \\
a \lor (b \land c) \lor (\sim a \land b) \lor (c \land b) & \leftrightarrow \\
a \lor (b \land c) \lor (\sim a \land b) \lor (b \land c) & \leftrightarrow \\
a \lor (\sim a \land b) \lor (b \land c) & \leftrightarrow \\
(a \lor (\sim a \land b)) \land (a \lor b) & \leftrightarrow \\
(a \lor (\sim a \land b)) \land (b \land c) & \leftrightarrow \\
I \land (a \lor b) & \leftrightarrow \\
a \lor b & \leftrightarrow (b \land c) \\
a \lor b & \leftrightarrow 
\end{align*}
\]

**RULES USED**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>[(p \lor q) \land r \leftrightarrow (p \land r) \lor (q \land r)]</td>
</tr>
<tr>
<td>03</td>
<td>[p \land q \leftrightarrow q \land p]</td>
</tr>
<tr>
<td>06</td>
<td>[p \lor q \leftrightarrow q \lor p]</td>
</tr>
<tr>
<td>02</td>
<td>[p \lor p \leftrightarrow p]</td>
</tr>
<tr>
<td>05</td>
<td>[p \lor (q \land r) \leftrightarrow (p \lor q) \land (q \lor r)]</td>
</tr>
<tr>
<td>01</td>
<td>[p \lor \sim p \leftrightarrow I]</td>
</tr>
<tr>
<td>07</td>
<td>[I \land p \leftrightarrow p]</td>
</tr>
<tr>
<td>04</td>
<td>[p \lor (p \land q) \leftrightarrow p]</td>
</tr>
</tbody>
</table>

Figure 7. Simplification of a switching circuit using Boolean Algebra.
Hypothesis: \(\angle EBA \& \angle BAD, \angle FDA \& \angle BAD, AB = AD, EB = FD\)

Prove: \(\angle EAB \cong \angle FAD\)

**KEY:**

"&" means "is supplementary to"

"hyp" means "hypothesis"

"ri" means "replace in"

"conj" means "conjoined with"

<table>
<thead>
<tr>
<th>STATEMENTS</th>
<th>REASONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. (AB = AD)</td>
<td>hyp</td>
</tr>
<tr>
<td>2. (EB = FD)</td>
<td>hyp</td>
</tr>
<tr>
<td>3. (\angle EBA &amp; \angle BAD, \angle FDA &amp; \angle BAD)</td>
<td>hyp</td>
</tr>
<tr>
<td>4. (\angle EBA \cong \angle FDA)</td>
<td>3 ri Th 10</td>
</tr>
<tr>
<td>5. (AB = AD, \angle EBA \cong \angle FDA, EB = FD)</td>
<td>1 conj 4 conj 2</td>
</tr>
<tr>
<td>6. (\triangle ABE \cong \triangle ADF)</td>
<td>5 ri Th 11</td>
</tr>
<tr>
<td>7. (\angle ABE \cong \angle ADF, \angle EAB \cong \angle FAD, \angle BEA \cong \angle DFA)</td>
<td>6 ri Th 12</td>
</tr>
<tr>
<td>8. (\angle EAB \cong \angle FAD)</td>
<td>7 ri TL 20</td>
</tr>
</tbody>
</table>

**THEOREMS USED**

**Th 10.** \([\angle XYZ \& \angle UVW, \angle PQR \& \angle UVW] \rightarrow \angle XYZ \cong \angle PQR\)  
\([\angle EBA \& \angle BAD, \angle FDA \& \angle BAD] \rightarrow \angle EBA \cong \angle FDA\)

**Th 11.** \([UV = XY, \angle UVW \cong \angle XYZ, VW = YZ] \rightarrow \triangle UVW \cong \triangle XYZ\)  
\([AB = AD, \angle ABE \cong \angle ADF, BE = DF] \rightarrow \triangle ABE \cong \triangle ADF\)

**Th 12.** \(\triangle XYZ \cong \triangle UVW \rightarrow [\angle XYZ \cong \angle UVW, \angle ZXY \cong \angle WUV, \angle YZX \cong \angle VWU] \triangle ABE \cong \triangle ADF \rightarrow [\angle ABE \cong \angle ADF, \angle EAB \cong \angle FAD, \angle BEA \cong \angle DFA]\)

**TL 20.** \([p, q, r] \rightarrow q\)  
\([\angle ABE \cong \angle ADF, \angle EAB \cong \angle FAD, \angle BEA \cong \angle DFA] \rightarrow \angle EAB \cong \angle FAD\)

Figure 8. A proof of a theorem in geometry.
The most useful theorems in geometry are often extremely lengthy if stated fully in symbolic terms and some abbreviation would be desirable. For example, consider the theorem which is customarily stated in English as follows:

If two sides and the included angle of one triangle are congruent respectively to two sides and the included angle of another triangle, the two triangles are congruent.

An abbreviated statement of this theorem for use in a proof teaching logic might read as follows:

For each triangle UVW and for each triangle XYZ,

\((UV \cong XY, \angle UVW \cong \angle XYZ, VW \cong YZ) \Rightarrow \Delta UVW \cong \Delta XYZ.\)

This statement is nearly equivalent to the statement in English above, but to make it completely so would make it too complicated for convenient use.

The abbreviated statement implicitly includes the conditions that UV and VW must be sides of triangle UVW and that XY and YZ must be sides of triangle XYZ by virtue of the linking rule for substitution: each variable must be replaced by copies of the same expression. Together with the linking rule, which is a fundamental part of the PROOF program, the statement also implies the condition that each angle named be included between the two sides named from the same triangle. However, the linking rule precludes always maintaining alphabetic order in names. Thus, if the variable, ‘\(\angle XYZ\)’ is replaced by ‘\(\angle BAC\)’, the linking rule then requires that ‘XY’ be replaced by ‘BA’. Finding a solution to the multiple name problem is clearly required, for the linking rule is a fundamental principle of mathematical logic.

In order to use the above theorem in a proof constructed on a general problem-solving system, a complex series of commands is required. It would be necessary to arrange a conjunction of three statements in the proper order to form the antecedant (as in line 5 of Figure 8), and it would be necessary to specify the substitution for each of the six variables. This procedure is already a more complex task than seems pedagogically
desirable for geometry proofs. It is evident that the proof shown in Figure 8 is a fairly simple one, as far as geometry theorems go, and yet substitution is required in three theorems of the complexity of the one discussed above.

As has been mentioned previously, it has been shown that computers can be programmed to generate proofs of theorems such as might be required to justify shortcuts typically taken in algebraic problem solving. Programs of this sort might conceivably be developed to check the justifiability of the gaps that must needs be left in geometry proofs,6 if they are to be reasonably simple to construct on an automated problem-solving system. We do not contemplate immediately developing a teaching logic which has such capacities, consequently, extensive work in geometry may have to be postponed until our experience with simpler problems makes such a development seem worthwhile. The new features which we are planning for the next version of a mathematical problem-solving system are briefly described in the final section of this report.

6 For examples of such logical gaps, see Unit 6 of the UICSM text, High School Mathematics. Urbana: University of Illinois Press, 1960.
6. Major Features of the System which is in Preparation for Mathematical Problem Solving

Since completing the PROOF program, the authors have been developing a plan for a more general system. The new system is planned for operation with newly developed equipment (PLATO III) which will provide twenty or more student-stations, a larger and more flexible keyset, and certain other technical improvements in the plotting and erasing features.

The present plans will permit intricate branching. Specifically, they provide for the construction of the steps in an exercise by:

a. forming an instance of one of the hypotheses given in the exercise or of any principle given or assumed by the student;
b. choosing any prior step as a source to be transformed by the application of one of the stored principles, another line, or by an assumed transformation rule entered by the student;
c. conjoining two or three prior lines in accordance with the rules of logic governing the particular connective chosen;
d. forming a statement which conjoins transformations of two or three prior steps.

The new system will permit the student to formulate a lemma for use in working an exercise by "typing" its pattern-sentence. He may then prove the lemma in the same manner he uses to prove theorems given as exercises, or he may assume the lemma and use it in working any of the given exercises of that lesson. Unlike the pre-stored transformation rules, use of this lemma is restricted to the student who formulated it. Theorems which were proved using assumed but unproved lemmas are not accepted for use in other exercises; however, the student may defer proof of a lemma until he has finished proving the exercise theorem and still have his proof accepted.