DOCUMENT RESUME

ED 365 524 SE 053 907

AUTHOR	Clark, Kevin Andrew
TITLE	Constructing and Implementing Algorithms for the Teaching of Propositional Calculus by Computer.
PUB DATE	[91]
NOTE	35p.
PUB TYPE	Reports - Research/Technical (143)
EDRS PRICE	MF01/PC02 Plus Postage.
DESCRIPTORS	Algorithms; *Calculus; College Mathematics; *Computer Assisted Instruction; Computer Uses in Education; Higher Education; *Mathematical Logic; Mathematics Education; *Mathematics Instruction; Mathematics Skills; Programming; *Proof (Mathematics); *Skill Development
IDENTIFIERS	*Propositional Logic

ABSTRACT

The objectives of this research were to review existing computer-assisted instruction systems for propositional calculus proofs or elementary logic and to develop an instructional computer program that guides students in the valid construction of propositional calculus proofs. The system is unique in that it provides assistance at each step of the proof evaluation. This assistance exists in the form of correcting user input and giving hints that will lead to the correct evaluation of the proof. The program also has the ability to suggest the correct axiom to use in a line of proof. The program is able to evaluate any proof that uses the elementary rules of logic, such as logical equivalence, logical implication, and rules of inference. (Author)

CONSTRUCTING AND IMPLEMENTING ALGORITHMS FOR THE TEACHING OF PROPOSITIONAL CALCULUS BY COMPUTER

by

KEVIN ANDREW CLARK

ABSTRACT

The objectives of this research are to review existing computer-assisted instruction systems for propositional calculus proofs or elementary logic and to develop an instructional computer program that guides students in the valid construction of propositional calculus proofs.

The system is unique in that it provides assistance at each step of the proof evaluation. This assistance exists in the form of correcting user input and giving hints that will lead to the correct evaluation of the proof. Not only does the program provide logical assistance in the form of hints, but the program also has the ability to suggest the correct axiom to use in a proof line. The program is able to evaluate any logic proof that uses the elementary rules of logic; such as logical equivalence, logical implication, and rules of inference.

INTRODUCTION

When students enter college, many are vaguely familiar with mathematical proofs. Most of them first encounter proofs in their high school geometry class, and frequently it is not a pleasant experience. Students think that solving proofs is difficult because there is no exact formula or unique path to follow in order to arrive at a valid logical proof.

When students take discrete mathematics in college, they are confronted with the topic of propositional calculus. In propositional calculus, logic proofs are generated statement by statement using previous statements and a set of axioms. Because of the multitude of approaches that can be used in constructing a proof, students tend to get discouraged and frustrated. One cause of this frustration is the fact that a line in a proof can be logically correct, but in no way apply to the valid construction of the proof itself. Another frustrating aspect is that an early error can go undetected while the student continues to generate new statements which are invalid because of the earlier error. It would be helpful to the student to know that all

of the previous steps of the proof are valid before proceeding to the next step of the proof.

A logical statement whose value is either true or false is a proposition; propositions are either compound and simple. A compound proposition is a proposition that can be decomposed into other propositions. A simple proposition is a proposition in which no part of it can be reduced into another proposition. In propositional calculus, letters are used to denote simple propositions. These letters are combined with logical connectives to form compound logical statements. The logical connectives used in propositional calculus are: $\cap$ (and), $\cup$ (or), $\rightarrow$ (implication), $\neg$ (negation), and $\Leftrightarrow$ (equivalence). The operations performed on propositions make up most of the section of mathematical logic called propositional calculus. Propositional calculus describes only those logical inferences for which one does not use the internal logic structure of the elementary propositions (Ross & Wright, 1988).

The goal of this paper is to design a computer program that will help students acquire the skills necessary to construct proofs using propositional calculus. This goal is in direct contrast with theorem proving because the students themselves are developing

theorems by using their own methods and distinct paths. In theorem proving, the students would have to determine the correct algorithm to use in order to construct a valid predetermined logical proof.

The program receives given statements and a conclusion from the user. The user is then asked to enter logic statements, compound or simple, that will logically lead to the conclusion. Each logic statement is evaluated at each step to assure that it is correct before proceeding to the next step of the proof. This procedure is different from theorem provers because of three functions. One, the program allows students to construct a valid proof using any logical method or path. Secondly, the program monitors and validates each step of the proof construction as the user progresses. And lastly, the program gives feedback at each step, when an error occurs, the program gives the user information that is specific and helpful to the nature of that error.

The program created through this research is similar to a program created at Stanford University under the direction of Patrick Suppes. Both programs provide user flexibility in constructing proofs and error correction feedback.

This paper encompasses three major research areas: computer-assisted instruction, mathematical logic and propositional calculus. All of these areas have a multitude of published research literature, but the specific area of computer-assisted proof evaluators has a very limited amount of published research literature. Much of the published literature centers around the research and development of computer-assisted logic proof programs in the late 1960's and throughout the 1970's.

The majority of the significant research and development of computer-assisted materials that pertained to the evaluation of logic proofs was conducted at Stanford University. A pioneer in the field of computer-assisted instruction and the leader of many of the research efforts in this area was Patrick Suppes of Stanford University. In 1963, the Institute for Mathematical Studies in the Social Sciences was established at Stanford University under the direction of Patrick Suppes and Richard Atkinson (Suppes, 1981). The institute started a program of research and development in the area of computer-assisted instruction, which later dealt specifically with computer-assisted proof evaluating. Although there was an abundance of

literature detailing the research efforts during the 1960's and 1970's, there was very little literature in the 1980's and 1990's.

In the literature concerning the implementation of computer-assisted mathematical logic programs, credence is given to the notion that logic as a mathematical concept can be taught in the early years of schooling. According to Inhelder and Piaget, children begin to develop a capacity for hypothetical reasoning by the age of eleven (Suppes & Binford, 1965). These findings have been used by educators to upgrade the mastery of mathematics in students. In the Suppes and Binford article, there are three ways in which educators try to upgrade the students mastery of mathematics: 1) they revised the high school curriculum to include calculus and analytical geometry. 2) they raised the level of mathematics achievement prior to entering high school. 3) they started teaching deductive techniques to gifted elementary school children (Suppes & Binford, 1965). Because of these findings, Suppes introduced the possibility of teaching logic at the elementary and high school levels. Suppes says that by exposing students to mathematical logic earlier it may help them develop better mathematical skills (Suppes & Binford, 1965).

In December of 1963, the first operational instructional program available to students dealt with elementary logic. The program was geared toward high achieving sixth grade students and was later expanded to include high achieving fifth grade students. The purpose of this instructional program is to guide students through the valid construction of propositional calculus proofs.

One of the most important aspects of this instructional propositional calculus program was that it accepted any logically valid statement. Because of this feature, the student was not restricted to a limited number of solutions nor were they restricted to a unique solution. In an experiment in which elementary school children were taught mathematical logic, Suppes showed that the upper 25% of the students could master conceptual and technical aspects of mathematical logic at a level of 85-90% achieved by comparable college students (Suppes & Binford, 1965). Furthermore, the goal of this experiment was to deepen and extend the mathematical experience of the elementary students to the broadest level of mathematics which pertains to methodology and proof theory. Even more specific, Suppes hoped that the experiment would teach

mathematical logic to fifth and sixth graders (Suppes & Binford, 1965).

In the academic year 1969-70, researchers at Stanford University used the experience gained from the first operational instructional program to prepare and test an introductory college course in elementary mathematical theories. The purpose of this course was to familiarize students with the theory of logical inference that emphasizes proving theorems. The course was taught solely by computer and the students were encouraged to try different paths or methods of proving theorems. At this time, the focus of research concerning computer-assisted instruction of mathematical logic slowly turned from pre-secondary applications to post-secondary applications. In an article by Goldberg and Suppes, they discuss an instructional program (which is written in Lisp) that is used to teach mathematical logic to college students (Goldberg & Suppes, 1974). The program determined whether the student had completed an exercise, provided a complete report on the axioms selected and which axioms and lemmas were needed to prove the theorem. This program covered topics such as: indirect proofs, truth tables, identities, axioms, theorems, translation, quantifiers, interpretation and

finding axioms. This instructional program was the only type of instruction used to teach the course. No human interaction was used except for graduate students who answered questions about the system.

There are some distinct differences between the program that was created in this research and the program that was created at Stanford University. This program is more user friendly (i.e., using formats and symbols that are familiar to its users). Also this program evaluates proofs using a truth table in addition to a set of axioms. Axioms and propositional statements are used to construct proofs using any logically valid method. In short, the two programs have similar objectives but different methods of achieving them.

IMPLEMENTATION

The implementation of this program encompasses three areas: presentation, logic, and evaluation. In the presentation portion, the program uses normal logical connectives and variables. The user is given the flexibility of using any letter as a variable while the program itself uses the variables p,q,r and s. Also imperative to the presentation of the program is the layout of the screen. The screen is organized such that the user has all of the pertinent information on the screen. In evaluating a proof by hand, most students use the conventional two column format. This format designates one column for the proof statement and the other column for the reference lines and axiom. This format is especially useful because it makes provisions for all of the information needed in the evaluation of the proof. It is because of the students familiarity with this format and its concise presentation of the information that this program emulates the two column proof evaluation format.

The logic aspect of the program implementation gives the student more flexibility in the construction of

the proof and it makes it possible for the program to provide specific feedback for logic errors and syntactic mistakes. The logic of this program is handled in four ways. First, logic statements are compared to determine their logical value. Then, the number and type of operators are determined. Both the type and number of operators must correspond to that of the other logic statement being used in the logical comparison. Thirdly, the order of the variables is obtained. The order of the variables in one statement must match exactly with the order of variables in another statement except in cases of commutation. Lastly, an exact match can be performed to determine whether the conclusion matches the final statement of the proof, thereby, validating the proof.

The final aspect in the implementation of the program is the evaluation. Once the logical value of a statement has been determined, the program must then evaluate the statement and all the statements that apply to determine the validity of that entire step in the proof. If the step is invalid then feedback in given pertaining to the nature of the error and the program may provide assistance in correcting the error. Because of the way the program evaluates the logical statements, the

program gathers information that is very useful in providing the user with feedback that is relative to the error. Through the implementation of this system, the user is be able to evaluate proofs that have multiple lines, is able to make partial line substitutions, and is able to evaluate proofs using direct and indirect methods. While the user is evaluating a proof, the system guides them through the proof informing them of any input mistakes or logical errors. In addition to informing the user of errors, the system provides useful information that is helpful in the valid construction of the proof.

Within the confines of this paper, the following terms must be defined: proof statement, reference line, axiom number and proof line. The proof statement is a logical proposition that is the result of previous proof statements and their application to a system axiom. The reference lines are previous proof lines that are referred to by a proof statement in order to create another proof statement. Axioms are assertions that are used without having to construct a proof. The proof line is the line of a proof that contains a proof statement, reference lines and an axiom. For example, in line 4 of Figure 1,

$\neg(\neg a \cup d)$, is the proof statement. The number 3 under the reasons column specifies the reference line used in the proof line and rule 21 implication specifies the axiom used in the proof line. All of the elements on line 4 make up the proof line. At the beginning of the program execution, the program receives preliminary input from the user which consists of the hypotheses, the conclusion and the type of proof that the user chooses to evaluate. The statements that are entered as the hypotheses are automatically listed as proof statements of the proof and have 'GIVEN' listed as their statement reason. If the user decides to evaluate an indirect proof, the negation of the conclusion automatically appears below the 'GIVEN' statements (e.g., as shown in Figure 2). If the user chooses to evaluate a direct proof then only the hypotheses are automatically written into the proof table (e.g., as shown in Figure 3). After the preliminary input has been received, the user enters the proof statement, reference lines and the axiom. Because the user can use any letter as a logical variable, the program must substitute its variables for the user's variables. This variable substitution occurs through two steps:

GIVEN: $a \rightarrow b, c \rightarrow d, \neg(a \rightarrow d)$ PROVE: $b \cap \neg c$

STATEMENTS	REASONS
1. $a \rightarrow b$	GIVEN
2. $c \rightarrow d$	GIVEN
3. $\neg(a \rightarrow d)$	GIVEN
4. $\neg(\neg a \cup d)$	3 rule 21 implication

Figure 1. The components of a proof line.

first, each substitute field that appears in the proof statement will be replaced by a unique character. The reason that the substitution fields are changed to unique characters is to eliminate inadvertent substitution. Inadvertent substitution occurs when a substitution is made that is correct according to the substitution procedure but incorrect according to the intent of the user. Once all of the substitutions are made, the system evaluates the proof line to determine its logical value and syntactic correctness.

One of the first stages in determining the logical value of a proof line is to evaluate at the reference lines. If the current proof statement has only one reference line, the logical correctness of the statement is determined in the following way: first the logical value of the left side of the axiom (which refers to the left side of the equivalence or implication operator) must be logically equivalent to the left side of the reference line. Both of these statements are logical steps that lead to the desired logical result. The logical results in this case are the right side of the axiom and the current proof statement. Secondly, the right side of the axiom must be identical, in both variables and operators, to the current

GIVEN: $a \rightarrow b, c \rightarrow d, \neg(a \rightarrow d)$ PROVE: $b \cap \neg c$

STATEMENTS	REASONS
1. $a \rightarrow b$	GIVEN
2. $c \rightarrow d$	GIVEN
3. $\neg(a \rightarrow d)$	GIVEN
4. $\neg(b \cap \neg c)$	Negation of conclusion

Figure 2. Indirect proof example.

GIVEN: $a \rightarrow b, c \rightarrow d, \neg(a \rightarrow d)$ PROVE: $b \cap \neg c$

STATEMENTS	REASONS
1. $a \rightarrow b$	GIVEN
2. $c \rightarrow d$	GIVEN
3. $\neg(a \rightarrow d)$	GIVEN

Figure 3. Direct proof example.

proof statement. This means that the two statements must have the same number of variables and operators appearing in the same order. An example of this can be shown in line 4 of figure 1. Given the substitutions for p and q at the bottom of the figure, line 3, $(a \rightarrow d)$, must be equal to the left side of the chosen axiom, $(p \rightarrow q)$. Line 4, $(\neg a \cup d)$, must be equal to the right side of the chosen axiom, $(\neg p \cup q)$. The negation symbol at the beginning of both statements in lines 3 and 4 is not evaluated because this is a partial line substitution. In the case of partial substitutions, much of the system implementations remain the same; the only difference is that the axiom only applies to part of the proof statement. When the system makes the necessary substitutions into the proof statement, it searches for the right side of the axiom in the proof statement. If the right side of the axiom appears in the proof statement, the statement is evaluated to be logically correct. The program evaluates every other part of the proof line as if it were a proof line with total substitution being applied.

The most important aspect of partial substitution is that the correct substitutions must be entered for the system variables p, q,r and s. The user must be certain that they only enter the part of the proof statement that

is relevant to the axiom and the proof statement. In the proof statement, the user enters the entire statement as it appears in the reference line. The user only changes the part of the statement where the axiom applies.

Only the variables and operators are considered in the evaluation of logical values for proof lines since parentheses do not change the logical value of a statement. If only one reference line is sited and the user inputs the substitutions for the system variables, the two statements should be identical. After substitution, both of these conditions also apply the right side of the rule and the proof statement of the current proof line.

If more than one reference line is sited, the system performs a logical conjunction of the reference lines and compares their logical value with the logical value of the left side of the axiom. Logically, this means that the logical value of all of the reference lines conjuncted together is equal to the logical value of the left side of the axiom. In addition, the operators used in the reference lines and in the left side of the axiom should appear the same number of times. This is necessary because a statement may have the same logical value as another statement, but the statement may use totally

different operators. In the case of multiple reference lines, we do not check for frequency or order of variables because there may be a variety of variables used within the different reference lines. Because of this, the program allows statements to be evaluated as being equal if the only difference is the order of the variables. For example, $(p \cap q)$ and $(q \cap p)$ may be evaluated as equal statements.

The logical value of a statement is obtained through the following algorithm. First, the statement is simplified from a compound logic statement into a simple logic statement. For example, in line 1 of the compound statement shown below, the statement itself is shown along with the levels of simplification that occur as the statement is being simplified.

1. $(p \cap ((q \rightarrow r) \cup s))$
2. $(p \cap a)$
3. $(b \cup s)$
4. $(q \rightarrow r)$

Line 2 is the first level of simplification, and in line 2, 'a' is substituted for $((q \rightarrow r) \cup s)$ in line 1. Line 3 is the second level of simplification, where 'b' is substituted

for $(q \rightarrow r)$. Finally, we reach the last line of simplification where the logical value of $(q \rightarrow r)$ is determined. The logical value of line 4 is substituted into line 3 as the variable 'b'. Then after the logical value of line 3 is obtained, it is substituted into line 2 as the variable 'a'. The logical value that is obtained in line 2 is the logical value of the entire compound logic statement. Once a compound statement is simplified, the logical value of the variables is obtained and the statement is evaluated according to the type of logic operator.

The variables take on values that are produced from all combinations of a four variable truth table. A truth table is generated for the givens, the conclusion, and each line of a proof to determine the logical correctness of the statement . After the value of the variable has been obtained, the logical operator generates the appropriate output value for each statement. For example, when the values of 'p' and 'q', in the statement $(p \cup q)$, are determined a logic operation is performed according to the 'or' operator. This means that the value of 'p' or 'q' must be equal to one (or a logical value of true). If this is not true then the value of the statement is evaluated to zero (or a logical value of false).

After the user has entered the reference lines, they may use the help key to view the program axioms. The unique aspect of this option is that the program will attempt to find the axiom that applies to the proof line and have it appear among the first five axioms displayed on the screen. In Figure 4 , the user has entered the proof statement, b, and the reference lines, 1 & 7 (after which the help key is pressed and the first set of axioms to appear on the screen are axiom 31 through 35). The axiom that applies to this proof line is axiom 31. There are instances when the program is unable to find the axiom that is applicable to the proof line. These instances are partial substitution, an incorrect logical statement and some cases of variable substitution. In these cases, the axioms appear on the screen starting with axiom number one.

Once the axiom is chosen, the user is given the option of symmetricizing the axiom if it is an equivalence relation. Symmetricizing allows the user to transpose sides to assure that the appropriate part of the axiom will be compared with the appropriate part of the proof statement and reference lines. Prior to symmetricizing the axiom, the axiom appears as it is in figure 5, $(p \rightarrow q) \Leftrightarrow (\neg p \cup q)$. After symmetricizing, the

axiom appears as it does in figure 6, $(\neg p \cup q) \Leftrightarrow (p \rightarrow q)$. After this step, the user must enter the substitutions for the system variables. The system will only prompt the user for those system variables that appear in the axiom. After the substitution step, the proof line is evaluated to determine if it is logically and syntactically correct. If the proof line is not valid, the user will receive a message pertaining to the nature of the error. If the proof line is valid, the user has the option to enter another proof line or to validate the entire proof.

To validate a proof means to show that the entire proof is correct and complete. The validation of a proof is accomplished by comparing the last proof statement and the user specified conclusion. For a direct proof, the final proof statement and the conclusion must be identical in the order and frequency of its operators and variables (e.g., as shown in line 12 of Figure 7). For an indirect proof, the last proof statement is evaluated; and if it is a contradiction the proof is validated. A contradiction is a statement that is always false (such as $(p \cap \neg p)$). In line 10 of figure 8, a contradiction exists because $\neg(a \rightarrow d)$ from line 3 and $(a \rightarrow d)$ from line 9 are both occurring within the same proof. Because this is

GIVEN: $a \rightarrow b, c \rightarrow d, \neg(a \rightarrow d)$ PROVE: $b \cap \neg c$

	STATEMENTS	REASONS
1.	$a \rightarrow b$	GIVEN
2.	$c \rightarrow d$	GIVEN
3.	$\neg(a \rightarrow d)$	GIVEN
4.	$\neg(\neg a \cup d)$	3 rule 21 implication
5.	$(\neg\neg a) \ll \neg d$	4 rule 18 DeMorgan
6.	$a \ll \neg d$	5 rule 1 double negation
7.	a	6 rule 29 simplification
8.	b	1,7

31	$(p \cap (p \rightarrow q)) \Rightarrow q$	modus ponens
32	$((p \rightarrow q) \cup \neg q) \Rightarrow \neg p$	modus tollens
33	$((p \cup q) \cap \neg p) \Rightarrow q$	disjunctive syllog
34	$p \Rightarrow (q \rightarrow (p \cap q))$	disjunctive special
35	$(p \leftrightarrow q) \cap (q \leftrightarrow r) \Rightarrow (p \leftrightarrow r)$	transitivity

Figure 4. The display of axioms.

GIVEN: $a \rightarrow b, c \rightarrow d, \neg(a \rightarrow d)$ PROVE: $b \cap \neg c$

STATEMENTS	REASONS
1. $a \rightarrow b$	GIVEN
2. $c \rightarrow d$	GIVEN
3. $\neg(a \rightarrow d)$	GIVEN
4. $\neg(\neg a \cup d)$	3 rule 21 implication

$(p \rightarrow q) \Leftrightarrow (\neg p \cup q)$ implication

Figure 5. Before the axiom is symmetricized

GIVEN: $a \rightarrow b, c \rightarrow d, \neg(a \rightarrow d)$ PROVE: $b \cap \neg c$

STATEMENTS	REASONS
1. $a \rightarrow b$	GIVEN
2. $c \rightarrow d$	GIVEN
3. $\neg(a \rightarrow d)$	GIVEN
4. $\neg(\neg a \cup d)$	3 rule 21 implication

$(\neg p \cup q) \Leftrightarrow (p \rightarrow q)$ implication

Figure 6. After the axiom is symmetricized

not possible, line 10 is said to be a contradiction. Since the proof includes the negation of the conclusion, the conclusion itself must be valid. The reason both proof validation methods focus on the last line of the proof is because all of the previous proof lines have been validated before proceeding to the next step in the proof. If the proof is not validated, the system prompts the user to enter the next proof line. The system continues to evaluate proof lines until the entire proof is validated.

GIVEN: $a \rightarrow b, c \rightarrow d, \neg(a \rightarrow d)$ PROVE: $b \cap \neg c$

	STATEMENTS	REASONS
1.	$a \rightarrow b$	GIVEN
2.	$c \rightarrow d$	GIVEN
3.	$\neg(a \rightarrow d)$	GIVEN
4.	$\neg(\neg a \cup d)$	3 rule 21 implication
5.	$(\neg\neg a) \cap \neg d$	4 rule 18 DeMorgan
6.	$a \cap \neg d$	5 rule 1 double negation
7.	a	6 rule 29 simplification
8.	b	1,7 rule 31 modus ponens
9.	$\neg d \cap a$	6 rule 4 commutative
10.	$\neg d$	9 rule 29 simplification
11.	$\neg c$	2,10 rule 32 modus tollens
12.	$b \cap \neg c$	8,11 rule 44 conjunction

THIS IS A VALID PROOF

Figure 7. Validation of a direct proof.

GIVEN: $a \rightarrow b$, $c \rightarrow d$, $\neg(a \rightarrow d)$ PROVE: $b \cap \neg c$

STATEMENTS	REASONS
1. $a \rightarrow b$	GIVEN
2. $c \rightarrow d$	GIVEN
3. $\neg(a \rightarrow d)$	GIVEN
4. $\neg(b \cap \neg c)$	Negation of conclusion
5. $\neg b \cup \neg\neg c$	4 rule 19 DeMorgan
6. $\neg b \cup c$	5 rule 1 double negation
7. $b \rightarrow c$	6 rule 21 implication
8. $b \rightarrow d$	2,7 rule 36 transitivity
9. $a \rightarrow d$	1,8 rule 36 transitivity
10. $\neg(a \rightarrow d) \cap (a \rightarrow d)$	3,9 rule 44 conjunction

THIS IS A VALID PROOF

Figure 8. Validation of an indirect proof

CONCLUSION

The goal of designing and developing an instructional computer program pertaining to the topic of propositional calculus proofs has been accomplished. In accordance with the original objectives, the system uses the two column format and provides students with feedback at each step of the proof construction. However, the parameters obtained during the evaluation of each line of the proof can be further used to enhance the amount of inferences and hints that the system provides to the user. In addition, the experience gained from using this program in the student environment will lead to a significant improvement in the pedagogical approach.

The concepts expressed through this research can also be applied to other aspects of propositional calculus and elementary logic, such as set theory and quantifiers. There exist software packages that deal with these topics. Discrete Math, which is produced by True Basic, is a computer program that deals with topics in elementary logic such as truth tables and Venn diagrams. Tarski's World, which is produced by the Center for the Study of Language and Information (CSLI), is a computer

program that deals with quantifiers and set theory. In addition to topics in propositional calculus, this program can be used as a vehicle of exploration in other topics such as geometric and mathematical proofs.

Finally, the data gathered through the implementation and use of this program can be analyzed to determine the effectiveness and validity of the claims set forth in this paper.

REFERENCES

Barwise, J. and Etchemendy, J. Tarski's World 3.0 (Software): Language of First-Order Logic. Center for the Study of Language and Information (CSLI). Stanford, California, 1990.

Goldberg, A. and Suppes, P. Computer-Assisted Instruction in Elementary Logic at the University Level. Educational Studies in Mathematics, 1976, Volume 6, pp. 447-474.

Goldberg, A. Design of a Computer-Tutor for Elementary Logic. Information Processing (IFIP), 1974, pp. 884-888.

Goldberg, A. and Suppes, P. Computer-Assisted Instruction in Elementary logic at the University Level. Institute for Mathematical Studies in the Social Sciences: Technical Report #239, November 1974, Stanford, California.

Kemeny and Kurtz. Discrete Math (Software). True Basic, Inc. West Lebanon, New Hampshire, 1988.

Larsen, I., Markosian, L. Z., Suppes, P. Performance Models of Undergraduate Students on Computer-Assisted Instruction in Elementary Logic. Instructional Science, 1978, Volume 7, pp. 15-35.

Ross, K. and Wright, C. Discrete Mathematics (2nd Edition). Prentice Hall. Englewoood Cliffs, New Jersey, 1988.

Stolyar, A. Introduction to Elementary Logic. MIT Press. Cambridge, Massachusetts, 1970.

Suppes, P. and Binford, F. Experimental Teaching of Mathematical Logic in the Elementary School. The Arithmetic Teacher, 1965, Volume 12, pp. 187-195.

Suppes, P. and Hill, S. Mathematical Logic for the Schools. The Arithmetic Teacher, 1962, Volume 9, pp. 396-399.

Suppes, P. and Hill, S. First Course in Mathematical Logic. Blaisdell Publishing Company. New York, New York, 1964.

Suppes, P. Computer-Assisted Instruction at Stanford. Man and Computer: Proceedings of the First International Conference on Man and Computer. Bordeaux, 1970.

Suppes, P. University-Level Computer-Assisted Instruction at Stanford: 1968-1980. Institute for Mathematical Studies in the Social Sciences, Stanford, California, 1981.