教育研究學報 1996，第十一卷第一期，24-31

Educational Research Journal 1996, Vol. 11, No. 1, pp. 24-31

Logex—
An Intelligent Computer Tutor in Logarithms

Fong-lok Lee

The Chinese University of Hong Kong

The use of computer in education started more than thirty years ago. In traditional computer assisted instruction systems, all responses have to be preplanned and implemented at the designing stage. The system builders must prespecify all available routes through the space of teaching possibilities. Every test, every decision, every branch leading to some remedial material and every exposition must be written in advance (Goodyear, 1991). When considering the number of decision points with their corresponding responses, the possible number of combinations, even for small tutoring systems, will be enormous. This prevents computer assisted systems from being used in larger subject areas. Recently, with the aid of knowledge representation techniques originating from artificial intelligence, human knowledge can be incorporated into computer systems. Based on the knowledge incorporated, computer systems can now make judgments on students' responses and decide on suitable responses to be made in real time. There is no need to pre-install all the possible routes and decision points. The computer system is thus smaller in a sense that less memory space is required on the hardware. Besides, the responses generated in real time can be more flexible and more adapted to students' needs. Logex is an example of such systems, called intelligent tutoring systems, which are designed to help students in simplifying logarithmic expressions by working through the simplification process with the students. This article describes its principles, structure and the ways by which it helps students.

電腦應用於教育上已有三十年的歷史。早期電腦輔助教學系統的設計，由於每一問題、學生的可能反應、與及隨後的輔助辦法等，都需要預先估計，然後設置於系統內。這種辦法，除了比較繁瑣外，亦需要較大的電腦記憶空間來貯藏這許多資料。所以對於那些需要廣闊學科知識的範疇來說，要使用這種方法來設計電腦輔助教學系統，存在著不少的困難。近年來，由於人工智慧的發展，已能將人類的知識析出，將之設置於電腦系統內。基於此內置的知識，電腦系統可以就學生的反應，即時判斷其對錯，然後提出適當的補救辦法。由於此種系統比較接近真人教師輔導學生辦法的關係，所以被稱為智慧型電腦輔助教學系統 (intelligent tutoring system)。智慧型電腦系統由於不需要預置所有可能反應及輔導辦法，需要的電腦記憶空間比較少，相應地可以處理的知識範圍亦比較廣闊。本文即介紹一個這樣的智慧型電腦輔助教學系統 Logex，其設計內容及幫助學生辦法。

The use of computers in education, particularly in instruction, has been in existence for more than thirty years. The goal of computer-assisted instruction (CAI) is individualized instruction so that each student can be instructed differently according to his or her level of expertise as judged by the computer. In earlier approaches to such instruction programs using conventional programming techniques, the system builders must prespecify all available routes through the space of teaching possibilities. Every test, every decision, every branch to some remedial material and every exposition must be written in advance (Goodyear, 1991). This works fine for simple programs, but when instructions become more complex, a combinatorial problem arises: the number of decision points, branches and remedial materials would be so large that either it would be difficult to input them into any machine or the manpower involved would be tremendous.

A second approach to the problem is the designing of systems called Intelligent Tutoring Systems (ITS) or Intelligent Computer Assisted Instruction Systems(ICAI). The word 'intelligent' is used to denote that this kind of work, when done by a human, is considered as intelligent (Self, 1988). ITS takes a completely different approach from CAI by simulating what a human tutor does during the instruction. When we observe what a human tutor does in a tutoring process, we can see that he does not necessarily use some prescribed sequence of rules. Rather, his procedures are either spontaneous reactions to his student's needs or are based on strategies which have proved effective. All these are based on the student's knowledge as inferred from his responses, as well as the tutor's subject and pedagogical knowledge. Human tutors do not always possess a distinct set of instructions for each situation. What they need

Correspondence concerning this article should be addressed to Fong-lok Lee, Department of Educational Psychology, Faculty of Education, the Chinese University of Hong Kong.

are the knowledge and the inference mechanisms that can generate the decision in real time. As pointed out by Self (1988), ICAI differs from CAI primarily in its focus on the representation of knowledge of the subject matter and of pedagogical knowledge.

As only the knowledge in the form of rules but not the routes and decision points are to be stored, simulating the human tutoring process has a further advantage of reducing the memory space required in the hardware. Hence, ITS can be used in more complex subject domains when compared with the traditional CAI.

Knowledge in ITS

To store knowledge into ITSs, several techniques originally from the field of artificial intelligence can be employed. This includes semantic net or frames (Woolf, 1987), production systems (Anderson, 1992; Anderson, Boyle, & Yost, 1985) and skill graph (Mao & Lin, 1992). Different types of knowledge are involved in an ITS. Examples are knowledge of the student (Baffes, 1996); knowledge of the domain (Giangrandi & Tasso, 1995) and knowledge of how to teach (Clancey, 1982; Marcke, 1992). It is commonly agreed that four sets of knowledge should be included (Rambally, 1986; Woolf, 1987; Park, 1991; Garito, 1991) though they may be named differently. The sets are:

- Domain knowledge: knowledge about the subject domain;
- Student model: knowledge about the student;
- Tutorial knowledge: knowledge about how to teach;
- Communication knowledge: knowledge about how to communicate with the learner through the computer.

Each set of knowledge refers to a different kind of knowledge that an ITS should have, though not all such systems would incorporate all of them. Also, the arrangement of each set of knowledge in a system may not be the same. Earlier systems may mix all kinds of knowledge together, while later systems may put them into separate modules (Park, 1991). The separation of knowledge into modules enables easy expansion of the knowledge base of the system.

What is Logex

Logex is a simple ITS that helps to diagnose and correct students' errors in doing logarithmic problems. It is considered simple in the sense that, for the time being, it works only in a narrow subject area. However, the design of Logex enables its easy expansion to larger subject areas without any structural modification. The only limitations perceived are the speed and demand on the hardware used, and most importantly, our understanding of the problem solving processes of human beings.

Basically, Logex is a simulation of a human tutor's tutoring process. The system does not work out the whole problem solving process in advance to obtain some models to act as criteria for the students' performances. Instead, it uses a "model tracing" methodology of tutoring (Reiser, Anderson, & Farrell, 1985). At each stage of the process, the system infers the learner's internal state by matching his output with the problem state generated by using ideal (correct) and buggy (incorrect) rules stored in the system. Instructions will then be given according to this inference. Ways to obtain the ideal and buggy rules will be discussed in later sections.

The inference is made possible by two major components in Logex: a knowledge base and an inferring mechanism. Just like other ITSs, the knowledge base of Logex consists of the four types of knowledge described above. They are expressed as sets of rules and are obtained through source materials such as text books and students' exercises. On the other hand, while the present system is developed by using the Prolog language, a common artificial intelligence language, the inherent inferring ability of this language makes the inferring mechanism of the present system possible.

How Logex works

The system starts with the computer screen divided into three parts, named Blackboard, Notebook and Exercise Book respectively. The Blackboard acts as a communication medium between the tutor and the student. It is mainly used to display problems and messages from the tutor to the student. The Notebook acts as a student notebook. In the present case, formulae to be used for solving the problems are displayed. Lastly, the Exercise Book is where the student works on the exercises. Fig. 1 shows the screen arrangement at the start of the system.

```
Question 1                         <Blackboard>

Simplify log(6)
--------------------------------------------------
<Exercise Book>              | <Note Book>
  log(6)                     | log(2)=0.301
= log(2*3)                   | log(3)=0.4771
= log(2)+log(3)              | log(7)=0.8451
= 0.301+0.4771               | log(10)=1
= 0.7781                     | log(100)=2
```

Fig. 1. Screen Arrangement of Logex

Problems given to the student are pre-arranged according to their levels of difficulty and presented to the student sequentially. Problems are presented both on the Blackboard and the Exercise Book and the student is then prompted to simplify the given expression by typing in consecutively new expressions which he thinks are simpler than the previous one. Each expression entered represents a step in the simplification process and will be checked by the computer. The computer responds by displaying on the Blackboard the message "correct" if the student works correctly, or displays hints to correct the errors. An expression is considered correct if it satisfies the following conditions:

(1) Correct syntax used: Logex recognises numbers, terms and symbols that should appear in logarithmic expressions such as "log(5)" ,"3", "100", "+", "-", " * ", " / ". Correct combination of the above will be accepted. Others are treated as illegal.

(2) A step leading to the correct solution: Logex includes a set of rules that would lead to the correct answer to the problem when applied. Before a new expression is entered into the system, Logex first stores up the old expression, the one already entered, or the given problem if no expression is entered. For the new expression, Logex checks whether it can be deduced from the old expression by using one of the correct rules. If yes, the expression will be accepted. Otherwise, it will be rejected.

Logex does not just reject incorrect inputs. Instead, based on the knowledge (mal-rules) incorporated in the system, Logex infers why the error happened and suggests possible ways to correct it. Detailed description of how this could be done will be discussed in later sections. However, the ability to infer is the factor that characterises Logex as an intelligent tutoring system as opposed to an ordinary computer assisted learning system.

With the constant checking after each input, the student is thus kept on the correct path until he reaches the answer. The next problem will then be given until all the problems are solved.

Structure of Logex

Logex consists of the knowledge, namely domain knowledge, student model, tutorial knowledge and the communication knowledge, that an ITS would normally possess. These different types of knowledge are clustered in four modules. The following briefly describes each module separately:

The Expert Module

An expert in an area should be familiar with the domain knowledge in that area. Hence, the expert module consists of the domain knowledge required to solve the problems. Two types of rules are included: the strategic rules and the axiomatic rules. A strategic rule describes the strategy that a student would use to tackle a problem, while an axiomatic rule describes the process involved in the actual tackling of the problem. The following example serves to illustrate this difference:

When a student is required to simplify the expression:

log 6

he or she might immediately respond by trying to factorize the number "6". However, the actual factorization of "6" is the use of related axioms which is quite different from the recognition that "6" has to be factorized. Hence, there are two processes involved: the first is the strategic rule which describes the recognition of the need to factorize when a certain pattern is seen; and secondly the axiomatic rule describing the process when used in related axioms to do the actual factorization. In other words, a strategic rule describes when to do something while an axiomatic rule describes how to something.

According to Lewis, Milson & Anderson (1987), simply learning how the axioms manipulate symbols may be easier than learning when to apply that axiom in service of problem solving. However, both strategic and axiomatic components of a skill must both be well learned if the skill is to be applied successfully in problem solving.

The advantage of separating strategy rules from axiomatic rules is that the tutor's cognitive load can be lightened if he can focus on the student's strategic decisions at some points and application of axiomatic knowledge at others separately (Lewis, Milson & Anderson, 1987). Strategic and axiomatic rules are thus extracted from mathematics text books and incorporated into the expert module of Logex. Table 1 shows some examples of both types of rules used.

Table 1
Examples of Strategic and Axiomatic Rules Used in Expert Module

Strategic Rules: Rules governing the strategies used.		
1	IF	a pattern log(X*Y) is observed
		AND the logarithm of the product of X and Y is not available,
	THEN	set a subgoal to distribute log to (X*Y)
2	IF	a pattern log(X/Y) is observed
		AND the logarithm of the ratio of X to Y is not available,
	THEN	set a subgoal to distribute log to (X/Y)
3	IF	the expression contains a component in the form log X,
	THEN	set a subgoal to factorize X.
4	IF	the expression contains a pattern log(X),
		AND the logarithm of X is not available,
	THEN	set a subgoal to write X as ratio of Y and Z,
		where logarithms of Y and Z are available.
6	IF	the expression is in the form logX+logY, etc.,
	THEN	set a subgoal to find the logs and then sum them up.
7	IF	the expression is in the form A+B, where A, B are real numbers
	THEN	set a subgoal to add them up.
Axiomatic Rules: Rules related to mathematical axioms:		
1	IF	an expression log(X*Y) is to be simplified
	THEN	write it as log X + log Y.
2	IF	an expression log(X/Y) is to be simplified
	THEN	write it as log X - log Y.
3	IF	an expression X is to be distributed among (Y+Z),
	THEN	write it as XY + XZ
4	IF	an expression X is to be distributed among (Y-Z),
	THEN	write it as XY - XZ

The Student Module

This module consists of the model of students' knowledge in the subject concerned. Two types of models, the overlay model and bug-identification model have been identified (Elsom-cook, 1988) in the past. The first one describes the student's knowledge as part of the expert's knowledge while the latter incorporates the student errors (bugs) in addition to their correct knowledge. Fig. 2 shows these two models of student knowledge with respect to expert knowledge.

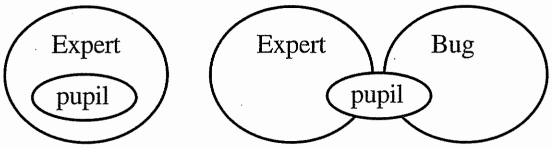

Fig. 2. Expert-based Modeling Method (Elsom-cook, 1988)

In Logex, the latter model is adopted since diagnosing is its major function and students' errors have to considered. Hence in addition to the strategic and axiomatic rules stored in the expert module, incorrect rules (called mal-rules) are also collected through students' exercises and included in the student model. Examples of mal-rules are shown in Table. 2.

Table 2.
Examples of Mal-rules Used in the Student Module

Strategic Mal-rules:		
1	IF	a pattern log(X+Y) is observed,
	THEN	treat it as log*(X+Y) and distribute log among (X+Y).
2	IF	a pattern log(X*Y) is observed,
	THEN	treat it as log*(X*Y) and distribute log among (X+Y).
Axiomatic Mal-rules: Incorrect versions of axiomatic rules used by students:		
1	IF	an expression X is to be distributed among (Y*Z),
	THEN	write it as XY * XZ
2	IF	an expression X is to be distributed among (Y/Z),
	THEN	write it as XY / XZ

The Tutoring Module

Logex includes knowledge obtained by observing how a tutor teaches a student in real situations and expresses them as rules. These rules indicate how the tutor reacts when a student error is encountered. Usually, this includes giving suitable feedback and asking the student to reenter. Table 3 shows examples of the rules used in Logex:

Table 3
Examples of Tutoring Rules Used in the Tutoring Module

Tutoring Rules: Rules for tutoring students		
1	IF	an integer is factorized incorrectly,
	THEN	display "wrong factorization" and let the student do it again.
2	IF	log(X*Y) is expressed as log X * log Y
THEN	display "Check Rule 1" and let the student do it again.	
3	IF	answer is obtained
	THEN	display "Congratulation", and let the student continue with next question if any.

The Communication Module

The communication module is responsible for handling the input and output of the system. The output component is relatively simple as this only involves displaying messages to the students. On the other hand, the input handling involves the complex task of checking and understanding of the texts entered by the students. This requires several artificial intelligence techniques and is handled by a component called Parser developed within the system. The Parser does two jobs: it checks whether the syntax of the text entered is legal and also converts the texts into codes understandable by Logex. Without the Parser, all interaction between Logex and the user is impossible.

The Language

The system was written with Prolog, an artificial intelligence computer language. The language system used is called Cogent Prolog (Amzi!, 1994). Prolog is a high level English-like programming language based on logical reasoning. It draws strength from the principles of mathematical logic - principles that were developed well before the invention of the computer (Walker, McCord, Sowa, & Wioson, 1987), and was recognized as the fifth generation computer language.

Knowledge involved in the four modules of Logex was written in the form of rules, or in the terminology of Prolog, the predicates or clauses. The inference mechanism embedded in Prolog will automatically control the whole tutoring process once triggered.

The Tutoring Process

Basically, the tutoring process is composed of the repetition of a small interaction cycle which involves the processes of prompting and waiting for students' responses, and giving feedback. Basically, a step done by the student may consist of one or more such cycles depending on whether he can be correct at the first attempt. Hence, for each question, several interaction cycles are repeated until the final solution is reached. The subsequent questions will then be given until the question bank is exhausted and the cycle is repeated until all the correct solutions are found. A flow chart showing how the tutoring process works can be found in Figure 3.

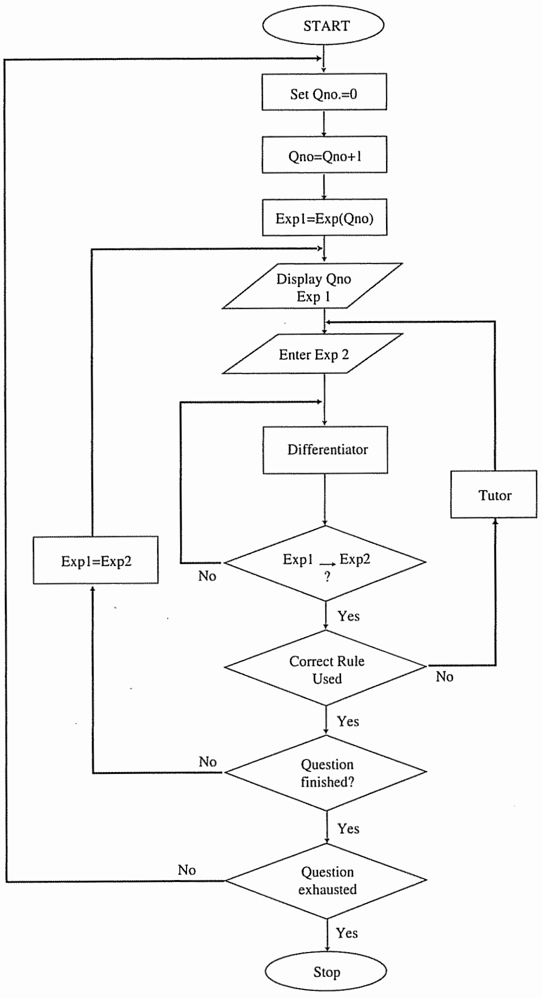

Fig. 3 Flow Chart Showing How Logex Works

The brain of Logex is the Differentiator which is in fact, doing all the checking and validating tasks. The Differentiator's job is to decide whether the entered expression is acceptable given the old expression, which is either the previously entered expression or the given question if it is at the beginning of a question. If the output from the Differentiator is "Yes", the student will be allowed to continue, either to the next step or the next question depending on whether the question is finished or not. If the output is "No", then the entered expression is considered not acceptable, and the student is asked to reenter by following the hints given by the computer tutor. Only when an acceptable expression is entered will the student be allowed to go on to the next step.

The Differentiator can be divided into four components: the Parser, the Arranger, the Generator and the Comparor. The Parser is responsible for parsing the expressions entered into forms recognizable by the computer. If the parsing process fails, the entered expression will be rejected. On the other hand, as an expression is made up of several elements including numbers, terms and operators, the order of these elements may vary even though the value or the meaning of the expression remains unchanged. Hence, for easy comparison between expressions, every expression entered has to be rearranged by the Arranger in a prespecified order.

The Generator is where most of the subject knowledge resides. The generator is responsible for generating new expressions from the old expressions based on the strategic and the axiomatic rules (either correct or incorrect) in the expert and student modules. The newly generated expressions will be compared with the expression entered by the student within the Comparor. If the two are matched, which means the computer can understand why the student entered the expression, the Differentiator will then send out a message to indicate a match can be found. This message may state that the expression entered is correct if the expression generated is induced with a correct rule, or if the latter is generated by an incorrect rule, the incorrect rule(s) used will be reported. Based on this message, the Differentiator can make suitable responses to the student.

If the expressions do not match, new expressions will be continuously generated until an identical one can be found. If finally, no match can be generated, which means that the computer cannot understand the entered expression, the student is prompted to enter again. Fig. 4 shows the relationship among the four components of the Differentiator and the process when an expression "log(2*3)" is entered against the old expression "log (6)" (in this case, it is the given question).

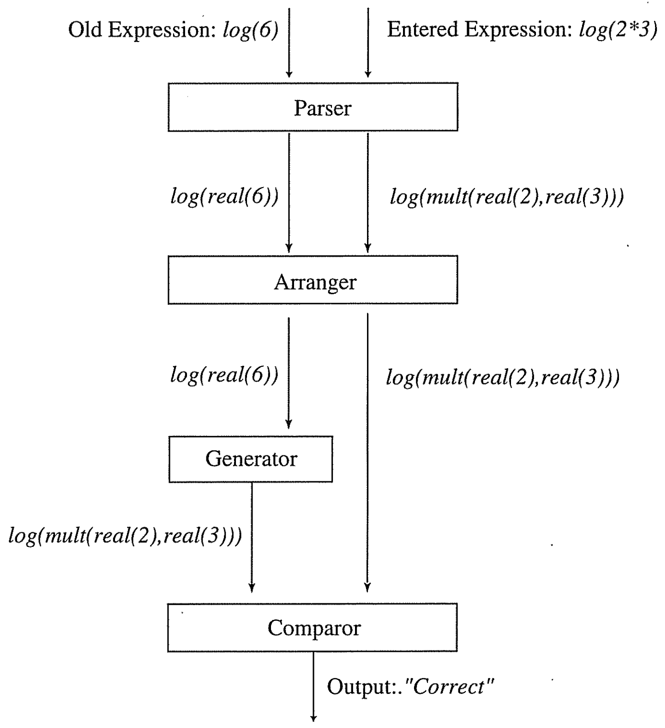
Old Expression: *log(6)*
Entered Expression: *log(2*3)*
Parser
log(real(6))
log(mult(real(2),real(3)))
Arranger
log(real(6))
log(mult(real(2),real(3)))
Generator
log(mult(real(2),real(3)))
Comparor
Output:. *"Correct"*

Fig. 4 Components of the Differentiator

Example

To show how Logex works, a simple example is given below:

Suppose the expression "log (6)" is to be simplified, the expression "log(6)" will be displayed both on the Blackboard and the Exercise Book. On the Exercise Book, below the given question, a "=" is also displayed and the student is then prompted to enter at the right-hand side of the equal sign.

If the student enters "log(2*3)", this newly entered expression will then be sent to the Parser and then to the Differentiator to check its validity and correctness (Fig. 4 shows the process). After checking with the existing correct and incorrect rules, the Differentiator returns the checking result. In this case, the result is correct. A "Correct" message will then be displayed on the Blackboard and the student is then prompted to continue with next line.

Suppose the student then enters log(2)*log(3) which is of course incorrect. The Differentiator finds that this can be generated by using the mal-rule "log(A*B)=log(A)*log(B)", hence a message showing why this is wrong will be displayed. In this case, the Blackboard displays "Check Rule 1", where rule 1 is displayed on the Notebook as "log(A*B)=log(A)+log(B)".

After the student reenters the correct expression, he will then be allowed to continue to type in further expressions such as "log(2)+log(3)". Logex checks every expression and returns suitable feedback according to the tutoring rules in the Tutor Module as shown in Table 3. The process is repeated until the final answer, in this case "0.7781", is reached. The message "Congratulation" will then be displayed on the Blackboard, and the student is allowed to proceed to the next question.

Conclusion

Logex demonstrates the possibility of using Artificial Intelligence techniques in developing computer tutors. Instead of prespecifying every response at every decision point, Logex simply incorporates the knowledge involved in the form of rules so that responses to students' inputs are generated in real time. This, in one way, greatly reduces the time involved in carefully planning every route at every decision point, and in another, saves much of the computer memory spaces. Both these factors make intelligent tutoring systems a practical tool to be used in real classroom situations.

Currently, Logex can only handle simplification problems in logarithm. However, it is designed so that without any structural modification, the system can be made to be used in larger subject domains by adding more knowledge. Hence, in principle, the system can handle similar tutoring tasks in any subject areas provided we can have large enough machines and sufficient understanding of human problem solving processes so that these processes can be recoded in the form of rules. Practically, if we restrict ourselves to some reasonably smaller subject areas, intelligent tutoring systems can be of great help to our students.

References

Amzi! (1994). *Manual of Cogent Prolog Ver.3.0.* Massachusetts: Amzi! Inc.

Anderson, J.R. (1992). Intelligent tutoring and high school mathematics. In Frasson, C., Gauthier G., & McCalla G. I. (Ed), *Proceedings, Intelligent tutoring systems, Second international conference, ITS '92.* (pp. 1-10). New York: Springer-Verlag.

Anderson, J.R., Boyle, C.E., & Yost, G. (1985). The geometry tutor. *Proceedings of the International Joint Conference on Artificial Intelligence.* (pp. 1-7). Los Angeles, CA.

Baffe, P. (1996). Refinement-based student modeling and automated bug library construction. *Journal of Artificial Intelligence in education 7* (1), 75-117.

Clancey, W. (1982). Tutoring rules for guiding a case method dialogue. In Sleeman D. & Brown J. S.(Ed). *Intelligent tutoring systems.* (pp. 201-225). New York: Academic Press.

Elsom-cook, M. (1988). Guided discovery tutoring and bounded user modeling. In Self J.(Ed), *Artificial intelligence and human learning.* (pp. 165-178). New York: Chapman and Hall.

Garito, M. A. (1991). Artificial intelligence in education: evaluation of the teaching-learning relationship. *British Journal of Educational Technology, 22* (1), 41-47.

Giangrandi, P., & Tasso, C. (1995). Truth maintenance techniques for modelling student's behaviour. *Journal of artificial intelligence in education, 6* (2/3), 153-202.

Goodyear, P. (1991). Research on teaching and the design of tutoring tutoring systems. In Goodyear. P (Ed). *Teaching knowledge and intelligent tutoring.* (pp. 3-23). NJ: Ablex.

Lewis, M. W., Milson, R. & Anderson, J. R. (1987). The teacher's apprentice: designing an intelligent authoring system for high school mathematics. In Kearsley, G. (Ed), *Artificial intelligence and instruction, applications and methods.* (pp. 269-301). Tokyo: Addision-Wesley.

Mao, Y & Lin, J. (1992). Intelligent tutoring system for symbolic calculating. In Frasson C., Gauthier G. & McCalla G. I. (Ed), *Proceedings, Intelligent tutoring systems, Second international conference, ITS '92.* (pp. 132-139). New York: Springer-Verlag.

Marcke, K. V. (1992). Instructional Expertise. In Frasson, C., Gauthier G., & McCalla G. I. (Ed), Proceedings, *Intelligent tutoring systems, Second international conference, ITS '92.* (pp. 234-243). New York: Springer-Verlag.

Park, O. (1991). Functional characteristics of intelligent computer-assisted instruction: Intelligent features. In The Educational Technology Anthology Series: *Expert Systems and Intelligent Computer-Aided Instruction.* (pp. 146-153). New Jersey: Eaglewood Cliffs.

Rambally, G. K. (1986). the AI approach to CAI. *The computer teacher, 13* (7), 39-42.

Reiser, B. J., Anderson, J. R., & Farell, R. G. (1985). Dynamic student modelling in an intelligent tutor for LISP programming. *Proceedings of the International Joint Conference on Artificial Intelligence* (pp. 8-14). Los Angeles, CA.

Self, J. (1988)(Ed). *Artificial intelligence and human learning.* Chapman and Hall.

Walker, A., McCord, M., Sowa, J. F., & Wilson, W. G. (1987). *Knowledge systems and Prolog.* Addison-Wesley.

Woolf, B. P. (1987). Theoretical frontiers in building a machine tutor. In G. Kearsley (Ed), *Artificial intelligence and instruction, applications and methods.* Addision-Wesley.