An experimental intelligent computer-based coach for teaching logic

Bumbaca, Federico

AN EXPERIMENTAL INTELLIGENT COMPUTER-BASED COACH FOR TEACHING LOGIC

Federico Bumbaca

Laboratory for Intelligent Systems
Division of Electrical Engineering
National Research Council
Ottawa, Canada

Introduction

Computer-based tutoring/coaching systems have the promise of enhancing the educational value of gaming environments by guiding a student's discovery learning. Games provide an enticing problem-solving environment that a student explores at will, free to create his own ideas of underlying structure and to invent his own strategies for utilizing his understanding of this structure. This paper is concerned with the tutorial resources required to 1) keep the student from forming grossly incorrect models of the underlying structure of the game, 2) help him see the limits of his strategy, and 3) help him discover the causes of errors.

The game is augmented by tutorial guidance that recognizes and explains weaknesses in the student's decisions by simply tracing its own embedded expert's reasoning processes. It also makes suggestions when the student asks for them. The coach is perceptive enough to make relevant comments but not so intrusive as to destroy the fun inherent in the game. For example, the student is only interrupted when playing at a suboptimal level. The degree of the intrusion is entirely dependent on the requirements of the student. The coach does not immediately point out the student's errors but allows him to examine his own behaviour and causes of his own mistakes. He may then seek additional assistance from the coach.

The Game

Cows and Bulls is played by two players, a Codemaker and a Codebreaker. The Codemaker selects a code which is concealed from the Codebreaker. The Codebreaker tries to guess the code and can obtain information about it in the following way.

A code is a sequence of N symbols selected from some set S. The Codebreaker submits a probe, which is also a sequence of N symbols from S. The Codemaker, the only player who sees both the code and the probe, tells the Codebreaker the score, which is a measure of similarity between code and probe whereupon the Codebreaker again submits a probe, and so on until the Codebreaker submits a probe equal to the code.

NRC No. 25579

The score consists of two components: the number of "cows" and the number of "bulls". A bull occurs for every location where probe and code have the same symbol. A cow occurs wherever among the remaining locations there is one in the code and one in the probe containing the same symbol.

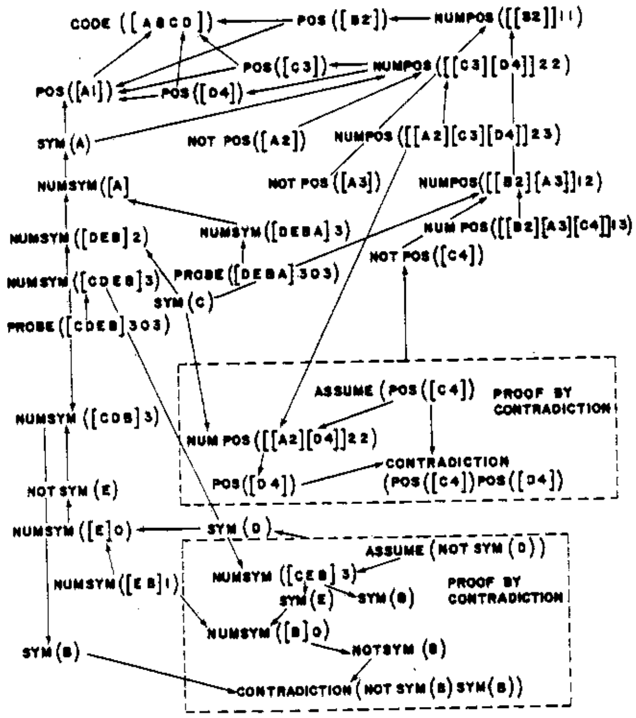
The System

The main components of the intelligent computer-assisted instruction (ICAI) system discussed in this paper are: 1) the natural language interface, 2) the domain expert, 3) the tutoring strategies, 4) the student model, and 5) the game-state-graph. The natural language interface must understand any questions or requests made by the user in a subset of English as well as explain any reasoning processes used by the system. The domain expert represents the problem solving expertise or knowledge that the system tries to impart to the student. It acts as the expert with which the student's performance is compared. This "differential" student model [1] [2] is then used in conjunction with the tutoring strategies to determine what to do next (ie. how to teach, when to break-in, what to say). The student model indicates what the student does and does not know, and the tutoring strategies specify how the system presents material to the student. All of these components are orchestrated in order to provide the student with instructionally effective advice. It is the game-state-graph which provides the medium through which these components interact. Self [3] is a classic discussion of the kinds of knowledge needed in a computer-based tutor.

The Game-State-Graph

The computer-based coach, during a game session, builds a state-graph which represents the current state of the game as well as all previous states. It is this graph which is used as the data-base for the student model, natural language interface, embedded domain expert, and tutoring strategies. It consists of two components: 1) system knowledge, and 2) system trace. The system knowledge includes all current known data with regards to the unknown code as determined by the embedded domain expert. The system trace consists of Horn clauses [4] whose conclusions make up the system knowledge and whose conditions have been previously proven or given. In simpler terminology, the trace explains how the system knowledge was derived. Sleeman and Hendley utilize a similar mechanism in their ACE system [2].

The domain expert builds this graph for use by the natural language interface, student model, and tutoring strategy modules. A segment of it is given in Figure 1 where it is noted that every assertion and hypothesis generated by the system is stored. This graph may be said to represent the perfect student in that he should be aware of this explicit knowledge derived from its implicit counterpart through the information given to him in previous probes and scores. It should be noted further that each clause represents a reasoning process which corresponds to a problem-solving step by a human problem-solver.



CODE (X)	=	X IS KNOWN TO BE THE CODE
POS ([S P])	=	SYMBOL S IS KNOWN TO BE IN POSITION P
SYM (S)	=	SYMBOL S IS KNOWN TO BE IN THE CODE
NUM SYM (X N)	=	N SYMBOLS IN THE LIST X ARE KNOWN TO BE IN THE CODE
NOTSYM (S)	=	SYMBOL S IS KNOWN NOT TO BE IN THE CODE
NOT POS ([S P])	=	SYMBOL S IS KNOWN NOT TO BE IN POSITION P
NUM POS (X B N)	=	N SYMBOLS IN THE LIST X ARE IN THE CODE OF WHICH B ARE IN THE CORRECT POSITION
PROBE (X C B N)	=	N SYMBOLS IN THE LIST X ARE IN THE CODE OF WHICH B ARE IN THE CORRECT POSITION
ASSUME (X)	=	ASSUME X TO BE TRUE
CONTRADICTION (X Y)	=	X AND Y CANNOT BOTH BE TRUE

Fig. 1 Segment of a Game-State-Graph

The Natural Language Interface

The natural language interface is based on an augmented transition network [5] utilizing a semantic grammar [6] as opposed to a strictly syntactic one [7]. It is based on the LIFER system of Hendrix et. al. [8] [9]. An application language is defined which is a subset of English that is appropriate for interacting with this particular program. This language specification is then used to interpret natural language inputs as questions or requests to the system.

The language is primarily specified in terms of grammatical rewrite rules or productions which specify the relations between certain strings of terminals and nonterminal symbols. These rules may be interpreted as simplified forms of augmented transition networks. Using these networks, the parser interprets inputs in the application language resulting in an interpretation in terms of the appropriate routines from the application system. The parser attempts to parse an input string from the top down and left to right by non-deterministically tracing down the network whose starting node is the start symbol. For example, suppose the following three production rules are defined as part of the application language:

```
<Sentence> -> is <SYMBOL> in <POSITION> | e1
 <Sentence> -> is <SYMBOL> in the code | e2
  <Sentence> -> is <PROBE> consistent | e3
```

If an input matches one of these patterns, the corresponding expression (e1, e2, or e3) is evaluated - these are the appropriate interpretations that the system is to make for the corresponding unit.

During parsing, the parser starts at the symbol Sentence and attempts to move toward the expressions to be evaluated at the right. The parser follows a branch only if some portion at the left of the remaining input string can be matched to the first symbol on the branch. Actual words can be matched only by themselves. Meta-symbols (such as <SYMBOL> or <PROBE>) can be matched in a number of ways:

1) As a simple set (eg. <SYMBOL> = the set {A B C D E F}).

2) As a predicate that is applied to the string to test for satisfaction (eg. a meta-symbol such as PROBE may be tested to determine if it consists of valid symbols as well as having the specified length).

3) By another network that has this meta-symbol at its starting node.

A large amount of semantic information is embedded in the synt ctic description of the application language. 'A and '3 are not defined as instances of the single meta-symbol <CHARACTER> as they would in a more formal grammar, but rather are separated into the semantic categories indicated by the meta-symbols <SYMBOL> and <POSITION>.

There are two additional features implemented to make interactions easier for the user - acceptance of grammatically incorrect inputs and a very limited form of ellipsis. Inputs such as "Is 'A in code?" and "'A in '3?", although grammatically incorrect are semantically unambiguous and are therefore accepted by the parser. This feature is implemented simply by extending the definition of the ATNs to include any sentences whose insignificant words are omitted.

A very limited form of ellipsis is dealt with by allowing the user to ask "Why?" after any system response. The system assumes that this question refers to the explanation of the previous response and therefore produces a further and more detailed explanation of the user's initial request. This mechanism allows the system to focus the interaction to as much detail as is required by the user. Once a satisfactory answer has been presented, the user may continue the dialogue at a higher level.

```
Welcome to Cows and Bulls
Length of Code?
4

List of Symbols?
[A B C D E F]


Probe?
[C D E B]


  Probes      Bulls Cows

[C D E B]       [0 3]   1

Probe?
[A D B E]


[A D B E] is inconsistent with previous Probes and Results!

Why?

This Probe is inconsistent because it is known that:
Symbol 'D is not in position '2

What is a valid probe?

The probe [F E D C] is consistent with previous Probes and Results

Is [D E B A] valid?

  Probes      Bulls Cows

[C D E B]       [0 3]   1
[D E B A]       [0 3]   2

Probe?
[E B A C]

  Probes      Bulls Cows

[C D E B]       [0 3]   1
[D E B A]       [0 3]   2
[E B A C]       [1 2]   3

Probe?
[B A C F]


[B A C F] is inconsistent with previous Probes and Results!

Why?

This Probe is inconsistent because it is known that:
Symbol 'D is in the code
Symbol 'F is not in the code
If Symbol 'B is in the code
        Then
                Symbol 'E is not in the code
                '3 symbols in the list [[B 2] [A 3] [C 4]] are in the code of
                   which '1 is in the correct position
If Symbol 'E is not in the code
        Then
                Symbol 'B is in the code
                '0 symbols in the list [E] are in the code
                '3 symbols in the list [[B 2] [A 3] [C 4]] are in the code of
                   which '1 is in the correct position
If Symbol 'A is in position '2
        Then
                '2 symbols in the list [[E 1] [B 2] [C 4]] are in the code of
                   which '1 is in the correct position
                Symbol 'C is not in position '2
If Symbol 'C is in position '3
        Then
                Symbol 'A is not in position '3
                '1 symbol in the list [[E 1] [B 2]] is in the code of which it
                   is in the correct position
                Symbol 'D is in position '4
'3 symbols in the list [[E 1] [B 2] [A 3] [C 4]] are in the code of which '1 is
   in the correct position

Which symbols are in code?

Symbol 'C is in the code
Symbol 'A is in the code
Symbol 'D is in the code

[E A C D]


  Probes      Bulls Cows

[C D E B]       [0 3]   1
[D E B A]       [0 3]   2
[E B A C]       [1 2]   3
[E A C D]       [2 1]   4

Probe?
[F B C D]

[F B C D] is inconsistent with previous Probes and Results!

Which symbols are not in the code?

Symbol 'F is not in the code
Symbol 'E is not in the code

Why is 'E not in code?

Symbol 'E is not in the code
        Because
                '0 symbols in the list [E] are in the code

Why?

'0 symbols in the list [E] are in the code
        Because
                Symbol 'D is in the code
                '1 symbol in the list [E D] is in the code

Why?

Symbol 'D is in the code
        Because
                Assume Symbol 'D is not in the code
                '3 symbols in the list [C E B] are in the code
                        Because
                                Symbol 'D is not in the code
                                '3 symbols in the list [C D E B] are in the code
                Symbol 'E is in the code
                        Because
                                '3 symbols in the list [C E B] are in the code
               Symbol 'B is in the code
                       Because
                               '3 symbols in the list [C E B] are in the code
               '0 symbols in the list [B] are in the code
                       Because
                               Symbol 'E is in the code
                               '1 symbol in the list [E B] is in the code
               Symbol 'B is not in the code
                       Because
                               '0 symbols in the list [B] are in the code
               The Following Assertions Are Contradictory:
                       Symbol 'B is in the code
                       Symbol 'B is not in the code

'1 symbol in the list [E D] is in the code
        Because
               Symbol 'A is in the code
               '2 symbols in the list [E A D] are in the code

Is 'B in '2?

Symbol 'B is in position '2

What is the position of 'A?

Symbol 'A is in position '1

Why?

Symbol 'A is in position '1
        Because
                Symbol 'B is in position '2
                Symbol 'C is in position '3
                Symbol 'D is in position '4
                Symbol 'A is in the code

[A B C D]

  Probes      Bulls Cows

[C D E B]       [0 3]   1
[D E B A]       [0 3]   2
[E B A C]       [1 2]   3
[E A C D]       [2 1]   4
[A B C D]       [4 0]   5

Congratulations, You Broke the Code!
```

Fig. 2 A Sample Session with the System

Upon completion of a successful parse, the database as defined by the game-state-graph is queried with the appropriate expression. The system then responds by evaluating this query and presenting its natural language representation to the user.

A simple language generator is utilized to implement the final stage of this interaction. Figure 2 illustrates a sample session with the system.

The Domain Expert

The domain expert component of this ICAI system is an articulate one [10]. It can explain each problem-solving decision in terms that correspond to those of a human problem-solver. Although it also consists of data structures and processing algorithms that do not necessarily mimic the reasoning steps of humans, it takes advantage of these algorithms only when the processing need be a "black-box" to the user. A typical example of such a situation may arise if the user were to ask the system for a suggestion as to a probe which is consistent with the known data. In such a case, faster and more efficient algorithms are utilized to produce a satisfactory response without the need to explain how this response was derived. This explanation is already resident in the game-state-graph in another form which the student is expected to acquire. The SOPHIE [11] and WEST [1] systems also take advantage of both articulate and black-box experts.

During a game session, after each acceptable probe is input by the user, the articulate expert is invoked to extract all possible data resulting from the score of this probe. This data is made explicit and stored in the game-state-graph for later use by the various system components. The reasoning processes within the expert are represented as production rules which the student is expected to eventually acquire.

These rules are predicate calculus well-formed formulas (wffs) [4] [12], some of which are listed in Figure 3. Formulas (a) to (c) represent the types of most inference rules in the system. Type (d), however, is significantly different in that it makes use of proof by contradiction. This rule is used only when all other rules fail and has proven to be extremely powerful.

(a) $(\forall x)(\forall z)(\forall n)\{\text{SYMBOL}(x) \land \text{NUMSYMBOL}(z\ n) \land \text{MEMBER}(x\ z)$

$\Longrightarrow [(\exists w)\{\text{SUBSET}(w\ z) \land \text{NOTMEMBER}(x\ w) \land \text{NUMSYMBOL}(w\ n{-}1)\}]\}$

(b) $(\forall x)\{\text{NUMSYMBOL}([x]\ 1) \Longrightarrow \text{SYMBOL}(x)\}$

(c) $(\forall x)\{\text{NUMSYMBOL}([x]\ 0) \Longrightarrow \text{NOTSYMBOL}(x)\}$

(d) $(\forall x)(\forall y)\{\text{ASSUMED\_PREDICATE}(x) \land \text{PROVEN\_PREDICATE}(y)$

$\land [(\exists z)\{\text{INFER}(z\ x \cup y) \land \text{MEMBER}(\sim z\ x \cup y)\}]$

$\Longrightarrow \sim x\}$

Fig. 3 Sample Inference Rules

Rule (a) states that if "x" is known to be a symbol in the code, "n" symbols in the list "z" are in the code, and "x" is a member of "z", then there exists a list "w" such that it is a subset of "z", "x" is not a member of "w", and "n-1" symbols in the list "w" are in the code. Rule (d) states that if "X" is a predicate assumed to be true (such as "A is a symbol in the code"), "Y" represents a subset of all predicates known to be true, and there exists at least one predicate "Z" which can be inferred from the union of "X" and "Y", and "ëZ" is a member of this union, then "ëX" must be true.

The expert's inference engine may operate in one of two domains: 1) it may use modus ponens [4] with the inference rules of Figure 3, or 2) it may first convert the wffs to clauses using a standard procedure [4] [12] and then use resolution [13] with unification [14]. Both are logically equivalent, although the latter method does not provide intuitively obvious inferences. The first method was implemented for this reason.

The production rule representation of domain knowledge allows the program to solve problems independently as well as criticize student solutions. Being able to solve the problems, ideally in any of several possible ways, is necessary if the ICAI program is to make fine-grained suggestions about the completion of partial solutions. However, currently the domain expert only solves a particular problem in one way. Once a solution has been generated, the inference mechanism stops and no longer searches for other possible solutions. This mechanism may easily be extended to deal with multiple solutions.

The Tutoring Strategies

The tutoring module of the coach must integrate knowledge about natural language dialogues, teaching methods, and the subject area. This is the module that communicates with the student, selecting problems for him to solve, monitoring and criticizing his performance and providing assistance upon request. The design of this module involves issues such as when it is appropriate to offer a hint or how far the student should be allowed to go down the wrong track. This additional knowledge, beyond the representation of the subject domain and the student's state of understanding (student model), is knowledge about teaching.

The teaching strategy employed is based on diagnostic modeling, in which the program debugs the student's understanding by evaluating his responses to the game [15] [16] [17] [18]. From the program's feedback, the student is expected to learn which skills he uses wrongly and which he does not use at all. This strategy is implemented through a coaching environment [10]. It is not concerned with covering a predetermined lesson plan within a fixed time. Rather, its goal is to encourage skill acquisition and general problem-solving abilities by engaging the student in the game of logic, Cows and Bulls. The immediate aim of the student is to have fun, and skill acquisition is an indirect consequence. Tutoring comes about when the computer coach, "observing" the student's play of the game, interrupts the student and offers new information in as much detail as is requested. A successful computer coach, such as WEST [1] and WUMPUS [19], must be able to discern what skills or knowledge the student might acquire, based on his playing style, and to judge effective ways to intercede in the game and offer advice.

The structure is in place for the implementation of more elaborate tutorial strategies. The current strategies which determine exactly when to intercede, what to say, and how to respond to student requests are quite simple. They do not deal with complex interactions which may dictate whether information should or should not be supplied to the user. For example, if the game-state-graph indicates that the code should be known by the user at a particular point in the game, it would be unwise to allow the user to ask the system for its value unless he is having extreme difficulty. If the system responded directly to each and every question by the student, he would not need to acquire any skills in order to play the game well. The strategies governing this interaction should only volunteer information if it is felt that the student would benefit from it. The major strategy actually implemented is one which is nonintrusive and rarely lectures. It will only interrupt the student when he is playing at a suboptimal level.

The tutoring strategies are implemented as production rules which may easily be altered, deleted, or appended to. Collins [15] has pioneered the careful investigation and articulation of teaching strategies. The WHY [20] and GUIDON [21] systems also explicitly articulate their strategies.

The Student Model

The student modeling module represents the student's understanding of the material to be taught. The purpose of modeling the student is to make hypotheses about his misconceptions and suboptimal performance strategies so that the tutoring module can point them out, indicate why they are wrong, and suggest corrections. It is advantageous for the system to be able to recognize alternative ways of solving problems including the incorrect methods that the student might use as a result of systematic misconceptions about the problem or inefficient strategies.

Since the student is primarily engaged in a gaming activity, any explicit diagnosing of a student's strengths and weaknesses must be nonobtrusive or subservant to his main activity. This means that the diagnostic component cannot use prestored tests or pose a lot of diagnostic questions to the student. Instead, the computer coach must restrict itself mainly to inferring a student's shortcomings from whatever he does in the context of playing the game. This can be a difficult problem. Just because a student does not use a certain skill while playing a game does not exist certain skill while playing a game does not mean that he does not know that skill. For example, a situation that required him to invoke it may never have been created. The absence of a manifested skill carries diagnostic value if and only if an expert in an equivalent situation would have used that skill. Hence, apart from the outright errors, the main window a computer-based coach has to a student's misconceptions is through a "differential" modeling technique that compares what the student is doing with what the expert would be doing in his place.

The process of constructing a differential model requires two tasks - both of which require the domain expert. The first task is evaluating the quality of the student's action (or probe) in relationship to the set of possible alternative moves that an expert might have made in the exact same circumstances. The second task is determining the underlying skills that went into the selection and composition of the student's move as well as each of the better moves of the expert. In this program, a complete model is not built since only the first task is carried out. The tutoring strategies utilize this model to guide the student - system interaction.

Conclusions

In general, ICAI programs have only begun to deal with the problems of representing and acquiring teaching expertise and of determining how this knowledge should be integrated with general principles of discourse. A successful computer-based coach has been developed which 1) tells the student whether or not his solution (probe) to a subproblem is correct, 2) identifies the source of an incorrect solution should one exist, and 3) explains why such a solution is incorrect in as much detail as is required by the user. Future work will focus on the development of a facility to deal with multiple solutions to a given subproblem, the incorporation of more complex tutoring strategies, and the development of a student model which explicitly determines the underlying skills that went into the selection and composition of the student's move.

Implementation Note

The program has been implemented on a Data General MV8000 using Waterloo Unix Prolog [22].

References

[1] Burton, R.R., Brown, J.S., "An Investigation of computer coaching for informal learning activities", Int. J. Man-Machine Studies, Vol.11, 1979, pp.5-24.

[2] Sleeman, D.H., Hendley, R.J., "ACE: A system which Analyzes Complex Explanations", Int. J. Man-Machine Studies, Vol.11, 1979, pp.125-144.

[3] Self, J.A., "Student models in computer-aided instruction", Int. J. Man-Machine Studies, Vol.6, 1974, pp.261-276.

[4] Kowalski, R., "Logic for Problem Solving", North Holland, Amsterdam, 1979.

[5] Woods, W.A., "Transition Network Grammars for Natural Language Analysis", Comm. ACM, Vol.13, No.11, Oct. 1970, pp.591-606.

[6] Burton, R.R., "Semantic grammar: An engineering technique for constructing natural language understanding systems", BBN Rep. No. 3453, Bolt, Beranek and Newman, Inc., Cambridge, MA, 1976.

[7] Winograd, T., "Language as a Cognitive Process, Volume 1: Syntax", Addison-Wesley, Reading, MA, 1982.

[8] Hendrix, G.G., Sacerdoti, E.D., Sagalowicz, D., Slocum, J., "Developing a Natural Language Interface to Complex Data", ACM Trans. Database Systems, Vol.3, No.2, June 1978, pp.105-147.

[9] Hendrix, G.G., "The LIFER Manual: A Guide a Building Practical Natural Language Interfaces", SRI International, Tech. Note 138, February 1977.

[10] Goldstein, I., "The computer as coach: An athletic paradigm for intellectual education", AI Memo 389, MIT, 1977.

[11] Brown, J.S., Burton, R.R., DeKleer J., "Pedagogical, natural language and knowledge engineering techniques in SOPHIE I, II and III", Int. J. Man-Machine Studies, Vol.11, 1979, pp.227-282.

[12] Nilsson, N.J., "Principles of Artificial Intelligence", Tioga Publishing Co., Palo Alto, CA, 1980.

[13] Robinson, J.A., "The Generalized Resolution Principle", Machine Intelligence 3, (Dale and Michie, Eds.), Oliver and Boyd, Edinburgh, 1968, pp.77-93.

[14] Robinson, J.A., "Computational Logic: the Unification Computation", Machine Intelligence 6, Edinburgh University Press, New York, (Meltzer and Michie, Eds.), 1971, pp.63-72.

[15] Collins, A., "Processes in acquiring knowledge", Schooling and the acquisition of knowledge (Anderson, Spiro, and Montague, Eds.), Hillsdale, NJ, 1976, pp.339-363.

[16] Brown, J.S., Burton, R.R., "Diagnostic models for procedural bugs in basic mathematical skills", Cognitive Science, Vol.2, 1978, pp.155-192.

[17] Brown, J.S., Burton, R.R., "Multiple representations of knowledge for tutorial reasoning", Representation and understanding: Studies in cognitive science, (Bobrow and Collins, Eds.), Academic Press, New York, 1978, pp.311-349.

[18] Koffman, E.B., Blount, S.E., "Artificial intelligence and automatic programming in CAI", Artificial Intelligence, Vol.6, 1975, pp.215-234.

[19] Goldstein, I., "The genetic graph: a representation for the evolution of procedural knowledge", Int. J. Man-Machine Studies, Vol.11, 1979, pp.51-77.

[20] Stevens, A., Collins, A., Goldin, S.E., "Misconceptions in student's understanding", Int. J. Man-Machine Studies, Vol.11, 1979, pp.145-156.

[21] Clancey, W.J., "Transfer of rule-based expertise through a tutorial dialogue", Rep. No. STAN-CS-769, Computer Science Dept., Stanford University 1979.

[22] Van Emden, M.H., Goebel, R., "Waterloo Unix Prolog User's Manual, Version 1.4", Logic Programming and Artificial Intelligence Group, Dept. Computer Science, University of Waterloo, 1985.