STUDENT MODELS: WHAT USE ARE THEY?

John Self, Centre for Research on Computers and Learning, University of Lancaster

ABSTRACT
The importance of student models to effective intelligent computer-aided instruction (ICAI) is often taken to be self-evident and as a result the precise role of student models tends not to be clearly defined. Rather than attempt to derive such a definition from educational principles, this paper reviews the actual uses which have been found for student models in existing ICAI systems. Twenty functions are identified in six categories: corrective, elaborative, strategic, diagnostic, predictive and evaluative.

1. INTRODUCTION

The standard view of ICAI systems is that they consist of three modules: the subject material, the student model and a teaching procedure (Clancey, 1986). Of these, it is the student model component which gives ICAI research its distinctive flavour, so much so that the development of student models has become an end in itself, it being taken as axiomatic that student models are essential to effective ICAI. Of course the student model is to be used by the teaching procedure - but how and why? We could try to answer this question by studying human tutoring expertise but I fear this will not be fruitful because human tutors use rather different student models (and perhaps not optimally). Instead, I propose to review the ICAI literature to identify the uses to which student models have actually been put, this to be regarded as a preliminary step to specifying when those uses are appropriate.

The student model is that part of an ICAI system which enables it to answer questions about the student using the system. Questions are broadly of four types: what can the student do, what does the student know about, what type of student is he or she, and what has the student done? So a student model may be considered a 4-tuple "P,C,T,H", where P describes procedural knowledge, C conceptual knowledge, T individual traits and H the history.

Typical members of P are procedures for performing multicolumn subtraction, symbolic integration or medical diagnosis. (These procedures may, of course, have bugs.) C might contain descriptions, expressed as frames, schema, networks or similar data structures, of the meaning of concepts such as prime number, metal or hepatitis. The distinction between P and C is not clear-cut. Taken together, K=P+C, they describe what the student knows. T is typically a set of labels, e.g. introvert, blind, bored, etc., describing the student. Some of the labels are assumed constant for the purposes of the ICAI system. H may be a transcript of the interactive session, summarised and interpreted to describe significant events.

An ICAI system is designed to develop what is described by K. The student model is maintained to aid this development.

2. FUNCTIONS OF STUDENT MODELS

The functions of student models may be classified into 6 types:
(1) corrective: to help eradicate bugs in what is described in K.

(2) elaborative: to help extend what is described by K (which may be considered 'correct' but 'incomplete').
(3) strategic: to help initiate more significant changes in tutorial strategy than the tactical decisions of (1) and (2) above.
(4) diagnostic: to help resolve the contents of K or what is described by K.
(5) predictive: to help determine the student's likely response to tutorial actions.
(6) evaluative: to help assess the student or ICAI system.

3. CORRECTIVE FUNCTIONS

If K is represented as a buggy variation B(K') of some desired knowledge K', then the student model may help in functions intended to eradicate the bug.

3.1 Bug identification

A pre-stored natural language description of B may be presented to the student:

> You multiplied the number of the multiplicand by the number directly beneath it in the multiplier and you wrote down the carried ignoring the units number. (MULT)

½** References for all systems mentioned are given in the Appendix.

This simply involves looking up B in a 'bug library'. The same message is given to all students when considered appropriate. LISP-ITS also associates a message with all buggy rules, but the message is a template to be instantiated to the problem-solving context and may provide a hint toward the correct solution, although neither of these features is illustrated in published examples:

> You should combine the first list and the second list,but LIST is not the right function ... LIST just wraps parens around its arguments.

The message depends (at most) on the present problem-solving context, i.e. on the contents of H, not those of K. The LISP-ITS messages are supposed to "describe why the answer is wrong".

PROUST makes a more serious attempt to relate messages to context. This is possible because it does in fact analyse the context (i.e. the complete program), unlike LISP-ITS. The general form of PROUST's messages seems to be `solution-specific statement" `general principle":

> You left out the initializations for the variables HIGHRAIN, ... Programs should not fetch values from uninitialized variables!
>
> You used a WHILE statement at line 19 where you should have used an IF. WHILE and IF are NOT equivalent in this context; using WHILE in place of IF can result in infinite loops.

As these examples demonstrate, the messages do not have to be a paraphrase or summary of the entry in the bug library (e.g. the second message is not directly related to the description of the 'while-for-if' bug). The entry may be regarded as a computational device for accessing a deeper misconception. The message associated with the entry can address the misconception rather than the bug. For example, in LMS, the message associated with the M1BRA2 mal-rule could discuss the commutativity of multiplication, rather than the mal-rule itself. PROUST's `general principle" is presumably intended to relate to possible misconceptions and to prevent mere patching of surface errors.

However, tying the message explicitly to the bug entry gives the potential advantage that the bug identification may be dynamically generated. This would

avoid the laborious pre-specification of messages and their resultant inflexibility. For example,

> Are you arguing then, that if you take any two places in the Northern Hemisphere, the one which is further north will have the lower average winter temperature? (WHY)

may be generated from an internal description of a buggy procedure for predicting temperatures. This technique is implicit in the description of WHY, but has not been implemented. A similar technique is conceivable for LISP-ITS's buggy rules. The technique is, of course, mandatory if B is produced by some generative theory of bugs, such as repair theory (Brown and vanLehn, 1970), rather than looked up in a library, since there is nothing with which to associate pre-specified messages.

In general, it is necessary for an element of P or C to be described meaningfully to a student. There are no programs which dynamically generate bug descriptions - consequently, there are none which generate descriptions as a function of K (to help make them more meaningful to the student), as ideally they should. For example, descriptions should not include concepts (e.g. multiplicand, perhaps) not known to the student, and should omit details assumed to be fully understood by the student. (Similar considerations arise with many following functions of student models).

3.2 Direct remediation

Sometimes the ICAI system offers a direct corrective statement, dealing with the knowledge that the student should have rather than the buggy knowledge he does in fact have. This happens in 4 situations:

(i) when the buggy knowledge is considered a mere factual error not indicative of a deeper misconception;
(ii) when the system is unable to determine the buggy knowledge but can at least determine which knowledge has been misused (for example, BIP can detect that required keywords are missing from a program and so informs the student of the constructs he should have used);
(iii) when the system cannot interpret a student's input but can itself determine what it should have been (for example, if LISP-ITS cannot find a buggy rule to match the student input, it uses the 'ideal rules' to determine the next step and so tells the student what he should have done);
(iv) when the more indirect tactics have been exhausted.

In general, the student model indicates that a particular knowledge item is missing or corrupted (but not necessarily how). The item is presented directly to the student, by methods similar to those in 3.1.

3.3 Indirect remediation

A variation of direct remediation is to provide only a summary or a part of the desired knowledge (i.e. a hint). For example:

> No, it is not useful to make that vertical angle inference here ... why don't you try to make an inference involving the fact that M is the midpoint of AB. (GEOMETRY)

3.4 Counterexample

A counterexample is a problem such that

çproblem¾ P çsolution¾
çproblem¾ P' çsolution'¾

where solution' (given by a correct procedure P') differs from the solution given by a buggy procedure P.

The problem can be generated by varying the problem parameters until the above conditions are satisfied but, in general, this is too inefficient. Alternatively, the buggy component of P may be identified and then, by regression analysis, the conditions which have to be satisfied by the problem in order that the buggy component be reached may be determined. The remainder of the problem definition may then be determined, at random, if necessary. The conditions could, of course, be attached directly to a description of the bug in a bug library (as in DEBUGGY), but as before, this is laborious and inflexible.

Counterexamples are of two types, depending on whether or not it is expected that the student will recognise that his 'solution' is incorrect. In the latter case, the counterexample is used as the opening gambit in an "entrapment strategy":

çstudent believes anywhere near the equator is very hot¾
Is it very hot on the Peruvian cost?
I don't know, but I guess so.
Well no, there is a very cold current coming up
the coast....
(WHY)

In the former case, we require (in addition to the above constraints) that K and solution be inconsistent,with the assumption that this inference will be made by the student. (This is Ohlsson's "obviously incorrect or impossible result").

çstudent believes you can grow rice anywhere hot¾
What about the Arizona desert?
çassumes student knows you can't grow rice in Arizona¾

A second example:

.(1 1 3 4 3 1
3 - - - = - - - = - 8 8 8 8 8
çstudent vulgarises by adding whole number to the numerator¾
1 4
3 - - - = ?
9 9
çassumes student knows that 0 is an incorrect result for subtracting two different fractions¾

In general, the problem of generating counterexamples for procedures is formally equivalent to the automatic generation of test data for programs.

3.5 Solution Tracing

Building the student model from the limited student input is an inductive process. If this process were sufficiently reliable, its result could be conveyed back to the student, to enable him to observe his own problem-solving processes 'in slow motion', and to reflect upon them. No ICAI system actually does this, but it might work as follows:

T: Solve 3x + 5 = 6
S: x = -2
T: Your solution steps were:
3x + 5 = 6
3 + x + 5 = 6
x + 8 = 6
x = -2

Of course, this function could be combined with 'direct remediation', by pinpointing the faulty step, or giving a correct trace.

3.6 Retrospection.

The student model may be used to address the source of difficulty, rather than the difficulty itself. For example, certain errors are known to arise from

'interference' between domains - e.g. between first and second language learning, between the two standard subtraction procedures, and between arithmetic and algebra. A pre-stored explanation of the error's derivation could be attached to the bug entry in the library, permitting a dialogue like:

T: Solve 3x + 5 = 6
S: 3 + x + 5 = 6
T: Excuse me,
$3\frac{1}{2}$ - is the same as $3+\frac{1}{2}$
but 3x is not the same as 3 + x.
In fact, 3x means 3 * x

However, any bug derivation which is sufficiently predictable to be worth including in the library could be addressed directly by instruction. What is really needed is the ability to dynamically analyse the history of interactions (recorded in H) using a model of the student's learning and mislearning processes (a component of P) to determine how a misconception could have arisen. This is the role proposed for the 'episodic memory' in the Lisp tutor of Weber et al (1987). Here, a faulty initialization in a function to compute a^n is explained by assuming that the student has drawn an analogy with, or has generalised from, the previous example, in which 1+2+..+n was computed, after initialising a result variable to 0. Unfortunately, no current learning models are able to perform this analysis.

A more straightforward function of H is to permit the system to refer back to previous successful uses of knowledge which the student appears to have forgotten. (No ICAI system seems to do this.) For example, in subtraction if a student makes a 'borrow across zero' error, refer back to a recent example in which a similiar (i.e. the preconditions for reaching the 'borrow across zero' code are satisfied) problem was successfully solved. To avoid extensive reanalyses, entries in H may be tagged with appropriate interpretations to permit such remindings.

3.7 'Try again'

The simplest corrective function that student models have is to help generate a problem of the same type for the student to 'try again'. This may be done by a similar analysis to that for counterexamples, or (e.g. LISP-ITS) having a bank of remedial problems which are considered to see whether they address the student difficulty.

3.8 Tactical withdrawal

Finally, the student model may be used to effect a withdrawal from the buggy area, by generating or selecting a problem which does not address the difficulty, or, better, does address some or all of the pre-requisite skills and knowledge, but no more. This may be appropriate when it is feared that the present difficulty may have de-stabilised previously acquired knowledge. (Surely some ICAI systems do this, but I have found no examples.)

4. ELABORATIVE FUNCTIONS

The elaborative functions are called on when K = K' - k, i.e. when the student's knowledge is considered 'correct' but incomplete. All the following subsections are concerned with the choice of the next topic: they differ in how the student model is used.

4.1 Curriculum-driven choice

If K and K' are mapped onto a pre-specified curriculum structure, defining the pre-requisite relationships between items of knowledge, then a new topic may be selected to meet some specified criterion. For example, a topic may be

selected which addresses only one node in K' which is not in K, and perhaps which maximises the number of nodes in K which are addressed. This is the basic strategy of BIP which selects that problem from a pool of 100 problems which 'optimises' the mixture of mastered and weak issues. Of course, problems could be dynamically generated (rather than selected), as discussed above.

Goldstein's genetic graph (Goldstein, 1982) is also based upon a "learning frontier" but with several refinements (none of which were followed through in any detail):

(a) the graph was supposed to also contain explicit 'deviation' links to buggy knowledge - a sort of compiled theory of bug generation - to help remediation;
(b) the links were to be labelled with information (e.g. analogy, specialization, etc.) which presumably was to help determine the mode of presentation of the topic selected;
(c) the student's 'learning preferences' were also to be overlaid on the links (see 5.2 below).
(d) intrinsic properties of the knowledge were to be related to the topology of the graph, e.g. isolated skills (i.e. ones with few links to other skills) might be considered more difficult to learn (see 4.3 below).
(e) the graph may be generated dynamically rather than provided as a static structure.

Even without these refinements, the use of a curriculum structure is a standard technique for determining topics. The technique is not, of course, completely deterministic - it depends on student performance (i.e. the contents of the student model) and perhaps on a random element, e.g. in selecting the particular node to address. But curriculum design is a laborious process and it does impose some, maybe too much, structure on the interactions. The following functions are less pre-determined.

4.2 Expert-student comparison

The choice of new topic may be determined by a direct comparison of K and K', without any reference to a curriculum other than any implicit in the structure of K'. For example, Kimball's INTEGRATION, which represents expert and student skill using two matrices giving the probabilities that a particular operator will be applied in a particular situation, determines the probable expert and student solution for potential problems, and then selects that problem which maximises the difference between the two solutions. Alternative selection strategies are conceivable (e.g. select the problem which gives the minimum non-zero difference) but the point is that they are not curriculum-driven.

Similarly, WEST's technique of discussing only issues which are used in a (covert) expert solution but not used in an actual student solution is independent of a curriculum. However, the technique is feasible only if the issues are essentially independent, with no prerequisite structure. In order to prevent the system referring to topics too far beyond the student's present understanding, LISP-ITS maintains not one "ideal model" but a series of them, handcrafted to stay in tune with a hidden curriculum. (There is an unimplemented suggestion that the ideal models may be created dynamically by the process of knowledge compilation.)

Other 'opportunistic' techniques leading to more complex tutorial decisions are discussed in section 5.1.

4.3 Internal analysis

The choice of new topic may be determined by an analysis of K alone, with no reference to any target knowledge K' or curriculum (just as one can offer debugging advice from analysing the program itself without knowing its specification). For example, we might detect structural deficiencies, e.g.

isolated nodes in a genetic graph, which we may seek to remedy. Similarly, we may look for potential redundancies or omissions in the structure.

More ambitiously, we could try to ensure that K satisfies any general principles which it is believed models should satisfy to promote understanding and learnability. For example, the system could determine whether the 'no function in structure' principle (de Kleer and Brown, 1984) is violated, i.e. by causal relationships not being locally defined, and, if it is, endeavour to remedy it, on the assumption that the principle is sound. In general, the principles (if any can be defined) measure the intrinsic quality of the model and indicate areas of weakness.

4.4 Learner control

Finally, we may dispense with K, as well as K' and the curriculum, leaving the choice of next topic to the student. The student model has no function here unless, as is usual, the options open to the student are constrained by the system to those which it considers appropriate in the context. So, for example, GUIDON and LISP-ITS present a menu of choices for the next step, determined from the student model and curriculum, as described above.

5. STRATEGIC FUNCTIONS

The previous functions reflect the fact that the student model and the effect of tutorial actions are uncertain, and so ICAI systems proceed by making moment-to-moment decisions. Nevertheless, these 'tactical' decisions must be seen as part of some more global, 'strategic' plan, usually implicit. The counterexample in the entrapment plan is a simple illustration. Some ICAI systems attempt to make these plans more explicit to permit the student model to initiate changes in these plans.

5.1 Change of plan

Whether or not a particular tutorial action is considered the next step of some unfolding plan or as the first step of a radically different plan is often a matter of opinion. But sometimes the student model does cause the unfolding plan to be suspended. For example, in LISP-ITS if the student makes two errors which do not match entries in the bug libraries, then the system reverts to a 'design mode' where it works through the algorithm with the student before returning to the original 'coding mode'.

In MENO-TUTOR these changes are initiated by explicit "strategic meta-rules", encoded much like production rules. For example, one of the six strategic meta-rules is:

```
If the present topic is complete
   and the tutor has little confidence in its
     assessment of the student's knowledge
then shift from a detailed examination of a single
     topic to a shallow examination of a number of
     topics.
```

Another approach is to allow these changes of plan to be provoked opportunistically. For example, GUIDON has a "related rules" procedure which interrupts the dialogue after a given rule has been discussed to present a related rule, i.e. a rule which makes the same conclusion and has some premise in common with the previous rule. The principle is, presumably, to associate other knowledge related to that just assimilated, so strengthening the knowledge structure.

5.2 Modus operandi

The second class of strategic function for student models is to cause changes in the global 'style' of tutorial interaction. Most of these changes depend on

the T (traits and characteristics) component of student models. The link between T and the changes is usually implicit and unprincipled.

Usually the T component is entirely static, i.e. depends only on prior assumptions about the student, and so the global style is predetermined. Only if T changes during the interaction will stylistic changes be initiated. The simplest dynamic change to T is made possible by asking the student to 'describe himself'. For example, WUSOR asks students to rate themselves on a four point scale and uses this information to turn off rules at the wrong level to avoid presenting inappropriate material. Similarly, one of GUIDON's rules says:

> If the goal of the consultation is being discussed
> and the student's estimation of his sophistication
> (on a scale from 0 to 4) is less than 3
>
> ...
>
> then show a sketch of the sub-goal tree for the goal currently being discussed.

This is not very interesting unless T is modified, as appropriate, by the system and the system knows what use to make of T. Usually, the modifications and the uses are superficial. For example, WEST's principle of only discussing unused issues on every other move (not at the earliest opportunity) is based on intuitions about the student's level of frustration, boredom, motivation, etc. and how this might be influenced by excessive interruptions. FLOW-TUTOR's use of timing information to determine when to intervene may perhaps be interpreted in terms of intuitions about how to deal with inattention or boredom.

These attempts to deal with 'personality' characteristics, of a stable or transient kind, are unsatisfactory because ICAI systems have insufficient information to estimate them reliably and because they can make little pedagogic use of them anyway.. More progress may be possible concerning 'cognitive' characteristics. MENO-TUTOR has some "tactical meta-rules" which may be thought of as attempts to make the tutorial style depend on the cognitive state of the learner, e.g.

> If the wrong answer threshold has been reached
> and the student seems confused then
> shift the discourse from an explicit correction
> and explanation of an incorrect answer to a
> simple response that recognizes, but does not dwell
> on, the incorrect answer .

WUSOR-II describes a student's learning abilities in terms of his "need for repetition, his degree of forgetfulness, and his receptivity to advice". It is not clear how these values are updated (if they are) and what pedagogic use is made of them (if any). It is also not clear if this is what is meant by the "learning preferences" to be overlaid onto the links of a genetic graph, nor how these relate to the conceptual relationships (abstraction, analogy, etc.) already linking nodes.

In short, no ICAI system makes significant, explicit pedagogic use of T. No ICAI system has an expressed aim to change what is modelled by T that is clearly related to its tutorial actions. ICAI systems do not deal directly with meta-level knowledge, i.e. knowledge about reasoning and learning strategies. The extent to which such knowledge is amenable to communication and modification by learners is an open question.

6. DIAGNOSTIC FUNCTIONS

The previous functions have tended to assume that the student model is sufficiently accurate, and that what is modelled is sufficiently clearcut, that

an appropriate action can be determined. This may not be the case in two situations: (a) if (to be specific) the student model indicates that the student knows P1 or P2 but the student model does not indicate which, and (b) if the student model indicates that the student knows that P1 or P2 obtains but the student is unsure which. The student model may be used to disambiguate such confusions before carrying out other functions.

6.1 Diagnose student model

DEBUGGY assumes that the student actually has a method for subtraction and that it is its task to isolate this method from the set of methods consistent with the student's performance so far. To do so it generates discriminating problems. With each bug is associated a "heuristic problem generator" which ensures that the generated problem fulfils certain conditions (e.g. it involves a carry). Candidate bugs generate a set of potential problems, from which one is chosen (if one exists) which gives different answers for different bugs. PIXIE avoids the need for a heuristic problem generator with each bug by symbolically executing mal-rules 'backwards' to generate a problem template. However, this only generates a problem to exercise a candidate bug: it does not generate a problem to discriminate between two candidate bugs. In fact, the required techniques are the same as for generating counterexamples (section 3.4) - the only difference here is that no one of the candidate procedures is assumed correct.

The ambiguity of the student model arises because the student input (his answer to a problem, say) is insufficient to allow the system to infer a unique interpretation of that input. Instead of generating more discriminating problems (as above), we could (a) ensure that all inputs are uniquely interpretable (LISP-ITS achieves this by arranging that the granularity of the rules is such that every character typed matches only one rule, at most), or (b) encourage the student to elaborate on previous inputs, to provide the necessary missing information. The MACSYMA ADVISOR attempts to carry out an 'interactive diagnosis' by asking the student about his beliefs which it tries to connect to assumptions associated with potential interpretations of the student's plan. Obviously, this is difficult in general because of the problems of natural language processing and the student's likely inability to discuss the items of interest (which is why the ADVISOR asks about beliefs and not the plans themselves).

6.2 Diagnose student

In this case the student model indicates that the student holds several hypotheses (with different degrees of belief) about the knowledge to be learned. There is no necessary implication that the student should not hold several hypotheses - he may not have had sufficient information to eliminate them. CTP assumes that the student does not have a single hypothesis about a concept to be learned but holds a set of them. It selects an example to present to the student which may help to discriminate between the hypotheses. The generation of a discriminating example involves the same techniques as in section 6.1 but the interaction with the student must be phrased differently.

7. PREDICTIVE FUNCTIONS

The student model may be used not only to analyse, interpret or explain past performance but also to predict performance and possibly learning, i.e. it may be run as a simulation of the student.

7.1 Performance prediction

Predicting the student's performance (at, say, solving a problem) is of little benefit if the prediction matches performance exactly. It is only useful if they differ and the prediction can be used to guide the analysis of performance

by limiting the search space of interpretations of the unpredicted data. For example, NEOMYCIN predicts the student's likely next steps and only if the step is unpredicted does it embark on a data-driven analysis. Similarly, LISP-ITS uses the ideal and buggy rules to predict possible next steps, and in fact does no data-driven analyses at all (unpredicted steps are left uninterpreted and after a threshold number of them the student is informed of the required next step). Prediction serves, then, to limit time-consuming data-driven analyses and to focus the analysis on what needs to be interpreted.

7.2 Learning prediction

If P includes a 'how to learn' procedure, i.e. it includes a model of the student's learning processes, then it may be used to predict the effect of didactic actions. Any decision to apply a didactic action (such as those above) is based on presumptions about the effect of the action. These presumptions are usually implicit and no more profound than, for example, FITS's assumption that if you tell the student something he has learned it. If the presumptions are made explicit, i.e. expressed as a procedure which, given a student model and didactic action as parameters, returns a new student model as output, then two benefits become possible. First, the system would have a basis from which to compute the actual new student model (only a basis since the learning prediction would not be entirely accurate), rather than have to compute it from scratch. Secondly, it would give an analytic reason for selecting a didactic action because we could (as in CTP) run the learning procedure with a set of potential actions and select that which is predicted to have the most beneficial effect, as determined by an evaluation measure defined over student models.

8. EVALUATIVE FUNCTIONS

The student model can be used to give an assessment or evaluation of the student and of the system, to some extent.

8.1 Student evaluation

In the simplest and most common case H can be used to give a summary of student performance (e.g. percentage correct). More interestingly, K can be used to describe what the student is thought to know, either with respect to a curriculum or in comparison to some target knowledge. The assessment may be presented to the student, a course administrator or the system itself (so that it can initialise the student model when the student re-uses the system).

8.2 System evaluation

Given a 'simulated student' (as in section 7.2), it is possible for the system to determine a predicted optimal teaching strategy. The same simulation could be used to predict the outcome of an alternative (say, random) teaching strategy. If the student model, including the 'how to learn' procedure, is sufficiently accurate then the difference between the predicted outcomes from the optimal strategy and from a computationally simpler alternative strategy gives a measure of the system's potential effectiveness, i.e. the prospect that it will provide significant improvements in learning. For example, assuming a particular Markov model for vocabulary learning, it can be shown that a decision-theoretic strategy provides a learning gain of 28% over a random strategy (Self, 1977b). These simulations in no way validate the student model and the learning procedure, but they give a basis for cost-efficiency decisions before actual implementation and experimentation with students.

9. CONCLUSIONS

Twenty different uses of student models have been identified from the ICAI literature, in an attempt to impose some order on what pragmatic designers have

developed. It is clear that there is a wide range of potential uses of student models, that in many cases the computational techniques are not well developed, and especially that the rationale justifying the various uses is often lacking. However, this catalogue of student model uses is only a beginning. We need to identify more carefully the contexts where the various uses are appropriate and to refine our techniques for carrying these functions out more effectively. This implies more empirical work on educational outcomes and more consideration of the relative merits of alternative computational representations.

APPENDIX

References for the ICAI systems mentioned in the text are listed here:

BIP - Barr, Beard and Atkinson (1976)
CTP - Self (1977a)
DEBUGGY - Burton (1982)
FITS - Woodroffe (1987)
GEOMETRY - Anderson, Boyle and Yost (1985)
GUIDON - Clancey (1983)
INTEGRATION - Kimball (1982)
LISP-ITS - Anderson and Reiser (1985)
LMS - Sleeman (1982)
MACSYMA ADVISOR - Genesereth (1982)
MENO-TUTOR - Soloway et al (1981)
MULT - Attisha (1983)
NEOMYCIN - Clancey and Letsinger (1984)
PIXIE - Sleeman (1987)
PROUST - Johnson and Soloway (1985)
WEST - Burton and Brown (1982)
WHY - Stevens and Collins (1977)
WUSOR - Goldstein (1982)

REFERENCES

Anderson, J.R., Boyle, C.F. and Yost, G. (1985), The geometry tutor, **Proc. Ninth International Joint Conf. on Artificial Intelligence**, Los Angeles.

Anderson, J.R. and Reiser, B.J. (1985), The Lisp tutor, **Byte**, 10, 4, 159-175.

Attisha, M.G. (1983), A microcomputer based tutoring system for self-improving and teaching techniques in arithmetic skills, **unpublished M.Sc. thesis**, Exeter University.

Barr, A., Beard, M. and Atkinson, R.C. (1976), The computer as a tutorial laboratory: the Stanford BIP project, **International Journal of Man-Machine Studies**, 8, 567-596.

Brown, J.S. and VanLehn, K. (1980), Repair theory: a generative theory of bugs in procedural skills, **Cognitive Science**, 4, 379-426.

Burton, R.R. (1982), Diagnosing bugs in a simple procedural skill, in D.H. Sleeman and J.S. Brown (eds.), **Intelligent Tutoring Systems**, New York: Academic Press.

Burton, R.R. and Brown, J.S. (1982), An investigation of computer coaching for informal learning activities, in D.H. Sleeman and Brown, J.S. (eds), **Intelligent Tutoring Systems**, New York: Academic Press.

Clancey, W.J. (1983), GUIDON, **Journal of Computer-Based Instruction**, 10, 8-15.

Clancey, W.J. (1986), Qualitative student models, **Annual Review of Computer Science**, 1, 381-450.

Clancey, W.J. and Letsinger, R. (1984), NEOMYCIN: reconfiguring a rule-based expert system for application to teaching, in W.J. Clancey and E.H. Shortliffe (eds.), **Readings in Medical Artificial Intelligence: the First Decade**, Reading, Mass.: Addison Wesley.

deKleer, J. and Brown, J.S. (1984), A qualitative physics based on confluences, **Artificial Intelligence**, 24, 7-83.

Genereseth, M.R. (1982), The role of plans in intelligent teaching systems, in D.H. Sleeman and J.S. Brown (eds.), **Intelligent Tutoring Systems**, New York: Academic Press.

Goldstein, I.P. (1982), The genetic graph: a representation for the evolution of procedural knowledge, in D.H. Sleeman and J.S. Brown (eds.), **Intelligent Tutoring Systems**, London: Academic Press.

Johnson, W.L. and Soloway, E. (1985), PROUST: an automatic debugger for Pascal programs, **Byte**, 10, 4, 179-190.

Kimball, R. (1982), A self-improving tutor for symbolic integration, in D.H. Sleeman and J.S. Brown (eds.), **Intelligent Tutoring Systems**, New York: Academic Press.

Self, J.A. (1977a), Concept teaching, **Artificial Intelligence**, 9, 197-221.

Self, J.A. (1977b), A state-space model for automatic instruction, **Computers and Education**, 1, 199-205.

Sleeman, D.H. (1982), Assessing aspects of competence in basic algebra, in D.H. Sleeman and J.S. Brown (eds.), **Intelligent Tutoring Systems**, New York: Academic Press.

Sleeman, D.H. (1987), PIXIE: a shell for developing intelligent tutoring systems, in R.W. Lawler and M. Yazdani (eds.), **Artificial Intelligence and Education**, Norwood: Ablex.

Soloway, E., Woolf, B., Rubin, E. and Barth, P. (1981), Meno-II: an intelligent tutoring system for novice programmers, **Proc. 7th International Joint Conference on Artificial Intelligence**, Vancouver.

Stevens, A.L. and Collins, A. (1977), The goal structure of a Socratic tutor, **BBN Report 3518**, Bolt Beranek and Newman, Cambridge, MA.

Weber, G., Waloszek, G. and Wender, K. (1987), The role of episodic memory in an intelligent tutoring system, in J.A. Self (ed.), **Intelligent Computer-Aided Instruction: Artificial Intelligence and Human Learning**. London: Chapman and Hall.

Woodroffe, M. (1987), Plan recognition and intelligent tutoring systems, in J.A. Self (ed.), **Intelligent Computer-Aided Instruction: Artificial Intelligence and Human Learning**. London: Chapman and Hall.