Student Modeling in ITS

Riichiro MIZOGUCHI
The Institute of Scientific and Industrial Research, Osaka University
8-1, Mihogaoka, Ibaraki, 567 Japan
miz@ei.sanken.osaka-u.ac.jp

Abstract: Student models play crucial roles in intelligent tutoring systems, since they provide useful information about what the students understand which enables the systems highly individualized tutoring. This tutorial paper is concerned with the student modeling methods proposed to date. Firstly, requirements to the student models are enumerated to give clearer understanding of various kinds of modeling methods proposed thus far. The requirements include executability, expressiveness of mal-rules as well as missing knowledge, coping with nonmonotonicity, etc. Secondly, modeling methods are discussed and characterized in terms of the requirements. Some examples of typical methods are described. The future direction is also discussed from practical and theoretical point of views.

1. Introduction

Education can be viewed as a process of intensive interaction between a teacher who has more knowledge and students who have less knowledge. An ITS: Intelligent Tutoring System is a system which realizes an adaptive and bi-directional interaction aiming at effective knowledge communication. By adaptive, I mean the system can give instruction appropriate to the individual student through dynamic control dependent on the learning process of the student. For this purpose, the system has to have a student model representing what he/she knows and what does not. It should also have a tutoring module which dynamically controls the course of instruction based on the student model. Thus, student model module, which provides useful information about the student with other functional modules in ITS, plays a crucial role in ITS. This tutorial paper is concerned with student modeling problems in ITS. The next section gives a rough overview of ITS. Basic viewpoints for student modeling are discussed in Section 3. In Section 4, typical methods are described together with comparison among them. Section 5 presents a brief explanation of a logic-based inductive modeling method.

2. Architecture of ITS

Typical ITSs are composed of three modules: student model module, expertise module and tutoring strategy module. Adaptability of the system depends largely on the information about the students, so the student model module plays a crucial role in ITS. A lot of efforts have been devoted to the problem of student model building, since it contains essential issues in ITS research such as what bugs are, where they come from, how a human solves problems, how deep the model should represent, in what language to represent the model, how to build it and so on.

The expertise module contains a teaching material represented in a declarative form. In other words, teaching material is stored as data of what to teach separately from the control information(how to teach) which is represented in the tutoring strategy module. This separation is one of the remarkable advantages of ITS over CAI in the sense of system architecture. This module is also responsible for evaluating the answers of the students and for producing explanation of the material itself and of the correct solution process.

To realize effective tutoring, the system has to decide what item to teach next, what problem to give next, how to correct the bugs and so on. Tutoring strategy module is responsible for these tasks. It mainly contains procedural knowledge about how to teach and acts as a scheduler.

Tutoring strategies in ITS are classified into two major categories such as direct instruction and indirect one. In direct instruction, explanation of an item under consideration and that of the solution or solution process are given along with explicit indication of the error the student committed. In indirect instruction, on the other hand, information given is made as implicit as possible so as to make the student notice his/her bugs by himself/herself. Guided discovery(Burton, & Brown, 1979) proposed by Burton and Brown is a typical example. In order to perform guided discovery, appropriate hints, suggestive counter-examples or analogical examples are given according to the information contained in the student model.

3. Issues on Student Modeling

This section discusses some basic issues on student modeling.

(1) Search space

First of all, we have to define the search space which is tightly coupled with the concept of bugs. One cannot model students without defining bugs which is one of the key concepts of student modeling. The definition of bugs in turn defines the search space for student modeling. From the computational point of view, bugs are defined as a mal-function or a mal-structure of the correct knowledge. By mal-function and mal-structure, I mean a state in which a component of knowledge does not realize its function and a state in which the structure of knowledge is incorrect, i.e., some component is missing or inserted, etc., respectively. From this point of view, there are three types of models which explain:

1) what component of the correct knowledge is incorrect(mal-function-1),

2) how the component is incorrect(mal-function-2), and

3) how the structure of the knowledge is incorrect(mal-structure).

Type 2 methods not only identify what component of the correct knowledge is incorrect in the student head but also model how it is incorrect in a limited way, e.g., using fault modes of each component. But it cannot model why the function is so incorrect. Needless to say, type 3 is the most powerful modeling, since it models what, how and why the component is incorrect by modeling the incorrect structure of the knowledge. Another issue is if the model reproduces the bugs or not. These issues determine the search space for student modeling.

(2) Multiple observations vs. single observation

In order to obtain a reliable model, it should be built by observing many data, though the didactic feedback becomes late. Therefore, there is a trade-off between obtaining reliable models and realization of quick and timely didactic feedback. Roughly speaking, analytic methods tend to model from a single observation, while inductive methods from multiple observations.

(3) Accuracy-cost trade-off

In general, the more accurate the student model becomes, the more effective the behavior of the system becomes. However, there exists a trade-off between accuracy of the model and cost to

build it. From a pragmatic viewpoint, we must set up an appropriate representation scheme for student models by taking the trade-off into considerations.(Self, 1988)

(4) Design with well-balanced needs and seeds

One sometimes finds an accurate model is over specification. Because student modeling is computationally expensive, we have to avoid to build over-specified student models. The role of a student model module is to provide a tutoring module with information necessary for generating adaptive behavior. There is no need to produce precise information more than required. Designers of student modeller always have to pay special attention to how to use the information contained in the model.

(5) Theoretical foundation

Many of the existing student modeling systems are domain-specific or requires a lot of domain-specific information. For this reason, it is difficult to discuss what are the common properties of the modeling methods developed and to accumulate the theoretical results. Domain-independent discussions and theoretical foundation for the student modeling mechanism are required. They contribute to both clarification of the inherent properties of the student modeling problem and to articulation of the scalability and reusability of the proposed mechanisms.

(6) Representation primitives

Every representation scheme requires primitives by which the objects are represented. Primitives in bug representation are extremely important. There are two kinds of primitives: 1) components of pre-enumerated bugs and 2) atomic components which can constitute any object in the material. By atomic, I mean there is no possibility of misunderstanding of the components. Modeling methods using the former approach have to analyze the student's bugs in advance which needs a considerably large effort, but the modeling is rather efficient. On the other hand, the methods based on the latter primitives can build a model in a bottom-up manner without any preparation of bug library. The grain size of the primitives is crucial in both cases.

(7) Inconsistency

Model building is done based on pairs of a problem and an answer to it. Most of the inductive methods build models from multiple data. Therefore, they are sensitive to inconsistency of the data. A set of data usually contains some inconsistent information due to the following reasons:

1) Students change their understanding as the learning proceeds.
2) Students sometimes make careless mistakes which are usually called slips.
3) Students sometimes have inconsistent knowledge in their heads.

The modeling methods are required to cope with this inconsistency.

4. Classification of Student Modeling Methods

We here classify typical modeling methods proposed to date and discuss the characteristics of them in this subsection (Wenger, 1987)(Dillenbourg & Self, 1992). Comparison of the methods discussed is summarized at the end of this section from the perspectives shown above.

4.1 Analytic methods

Student modeling is generally viewed as an inductive process in which a model explaining observed data is built. However, we can find another view, that is, analysis of the expertise model using the observed data(Hoppe, 92). A student model does not have to explain all the behavior of

the student. Considering that its function is to give necessary and sufficient information to the tutoring module, analytic methods work very well for many types of tutoring modules. By analytic methods, I mean those for obtaining information of the student's understanding state by analyzing the correct knowledge based on the student behaviors. Typical examples are overlay model(Carr & Goldstein, 1977), logic programming method(Hoppe, 1992)(Beller & Hoppe, 1993), and model-based cognitive diagnosis(Self, 1993). This type of methods do not model mal-structure.

(1) Overlay model(Carr & Goldstein, 1977)

Roughly speaking, knowledge the student has increases as the learning proceeds and eventually he/she comes to have the full set of knowledge that the teacher wants to teach. Overlay model is based on this simple idea and the model is represented as a subset of expertise model. Model building is easily done by marking the items he/she seems to understand which is the simplest method of analysis of the correct knowledge.

The search space is limited to within the correct knowledge. Although it only models mal-function-1, that is, what knowledge the student has and what does not by a simple mechanism, overlay model have been used in a lot of ITSs, since it is robust and easy to manipulate. The model is built from multiple data. Nevertheless, overlay model is robust to inconsistent data, since it does not manage any consistency or keep any dependency among the components of the model.

(2) Logic programming method(Hoppe, 1992)(Beller & Hoppe, 1993)

Hoppe proposes a method to analyze the correct knowledge represented as a Prolog program by executing the program with the student answer as a goal. When the student's answer is incorrect, the execution fails if it is interpreted by Prolog interpreter. In order to recover the execution failure, he introduces fail-safe meta-interpreter of Prolog and detects uncovered goals which potentially correspond to the student's bug. Many uncovered goals are usually detected, so he employs EBG(Explanation-Based Generalization) technique to formulate rules called error patterns which are used to select the goals corresponding to the plausible bug.

Search space of this method is small, since it is restricted to the computation tree of the correct knowledge. This method requires predefined classification rules which are roughly correspond to bug rules, though they are more abstract than bug rules. It represents bugs of type mal-function-2. Because this method builds model from one data, it does not suffer from inconsistency problem.

(3) Model-based cognitive diagnosis(Self, 1993)

Self proposes to use the GDE: General Diagnostic Engine developed by de Kleer for diagnostic problems in artificial intelligence(de Kleer & Williams, 1987). GDE was originally developed for troubleshooting of logic circuits and requires complete information of circuits topology. Knowledge of the many domains can be interpreted as a kind of logic circuits where functional modules such as AND/OR gates correspond to the operations used in the domain knowledge and circuit topology to the order of operation execution. GDE is logically based on an exhaustive search over the search space defined as a set of all the combinations of possible faulty components with the aid of ATMS which makes the search very efficient. So, the search mechanism, in principle, is to check what sets of components are faulty and to find out minimum set of faulty components. Another advantage of GDE is its automatic question generation mechanism for disambiguation of the fault hypotheses. Although it runs without bug library, it represents only mal-function-1. But a just small extension, that is, introduction of the fault modes into each component, enables it to cover mal-function-2 as Self indicates(Self, 1993). It is sensitive to inconsistent data.

Theoretical foundation becomes more firm and sound in this order, though there is a large gap

between overlay model and Hoppe's method. Hoppe's method and model-based method are nearly equally general and theoretically sound.

4.2 Inductive methods

While analytic methods do not model mal-structure, inductive modeling methods try to represent it. The issue here is how to define the search space for mal-structure. The approaches to this are roughly classified into two types according to how to define the search space. That is, the search spaces are defined using:

(1) Pre-enumerated bugs, or

(2) Primitive concepts or operations instead of pre-enumerated bugs.

The concept of mal-structure sometimes implies unexpected buggy knowledge which makes its modeling intractable. However, the reality is not so. Suppose a human teacher encounters a student with an unexpected buggy knowledge. He/she sometimes succeeds in modeling the student and sometimes fails according to his/her capability and the peculiarity of the bug. No human is perfect. We can model his/her activity as follows: That is, we can model mal-structure in the pre-determined search space, though the performance depends on the completeness of the search space. Therefore, the issue to discuss is not whether we can model it perfectly or not but how much complete the space is and how flexible and powerful the search mechanism is. The first type of methods search for mal-structure in the space defined by correct knowledge and pre-enumerated buggy knowledge and the second search in that defined by primitives selected.

The first type of methods are further divided into two subgroups:

(1-1) Enumerative modeling

(1-2) Constructive modeling

(1-1) Enumerative model

A typical example of this type is BUGGY model proposed by Brown et al.(Brown & Burton, 1978), where basic procedures in subtraction at elementary school level are represented in a network structure including correct and incorrect sub-procedures. Although it has greatly affected the succeeding research, the BUGGY model did not have a capability of building a model from examples.

(1-2) Reconstructive model

This type of models try to represent the bug by combination of pre-enumerated buggy concepts/operations and correct primitives. DEBUGGY(Burton, 1982), a successor of BUGGY, is augmented so as to build a model from examples. It has a lot of modular chunks of buggy sub-procedures rather than a network. DEBUGGY builds a model from a set of examples given at once, so it can be said to be an off-line type system. IDEBUGGY(Burton, 1982) is a successor of DEBUGGY, which builds a model in an interactive, that is, on-line way. It dynamically produces problems for building a model. Both systems have capability to cope with the noise problem, though it is rather limited. The method employed in DEBUGGY is called "coercions" which screen noisy data matching with pre-determined pattern of slips. IDEBUGGY produces another problem to identify whether the mismatch is caused by slips or not.

Similar method called LMS: Leeds Modeling System is proposed by Sleeman(Sleeman, 1983) which is a typical model representation method in terms of production rules. Let me present some examples of LMS in the algebra manipulation domain.

Rule name	conditions	Action
Solve	(SHD M*X = N)	(SHD X = N / M) or (SHD infinity)
NtoRHS	(lhs +|- M = rhs)	(lhs = rhs -|+ M)

M.Solve (SHD M*X = N) (SHD X = M / N) or (SHD infinity)
M.NtoRHS (lhs +|- M = rhs) (lhs = rhs +|- M),

where the first two are correct rules and the last two are mal-rules and SHD stands for "string head", "lhs" and "rhs:" for left-hand-side and right-hand-side(both can be null), respectively, and xly for x or y. Note that each mal-rule has corresponding correct one and has the same condition part as that of it but incorrect action part. LMS builds a student model in terms of a sequence of the correct- and mal-rules.

Search spaces of DEBUGGY, IDEBUGGY, and LMS are small, though it takes a lot of time and efforts to build a library of bugs or mal-rules. Depending on the completeness of the bug library, both methods can model mal-structure type of bugs. Theoretical foundation of them is rather shallow and they have no capability to cope with inconsistency.

(2) Models without the concepts of bugs

Three typical examples are given in the following.

1) Perturbation method

Bugs can be viewed as variants of the correct knowledge. So, they can be generated by applying perturbation operators to correct one. Perturbation methods are based on this idea. Takeuchi and Otsuki(Takeuchi & Otsuki, 1987) utilizes this type of method in their system in which two types of perturbation operators(domain-dependent and domain-independent ones) are introduced to augment the modeling power.

The student model in BOOK(Otsuki & Takeuchi, 1985) is based on perturbation model. The model contains a lot of information about the student besides the bugs. For example, it represents the corresponding correct knowledge from which the bug is derived, kinds of the perturbation applied, tutoring methods used for remedy and the degree of the student's understanding of the item.

The perturbation method is formulated as follows:

Find W such that

C - M + W |- A,

where C stands for the set of correct knowledge, W the incorrect knowledge obtained by perturbation of the correct knowledge M, and A behavior of the student.

BOOK has several perturbation operators independent of the teaching materials. Some typical examples are operators for deletion of a subproblem, mis-ordering of the subproblems and mis-use of similar methods. One of the advantages of the method is that it eliminates the need to prepare bug library beforehand. BOOK stores the bugs identified during instruction and builds another model as an average student model by statistical processing of the bugs.

The search space of this method is dependent on the complexity of the perturbation operators used. But, usually it is not so large. It has no firm theoretical foundation for it but it does not suffer from inconsistency of the data, since it builds a model from one observation.

As discussed above, building a student model is viewed as inductive inference from a set of examples. Essentially, there is no need to use pre-enumerated bugs to model students. What we need are primitive concepts in terms of which we can build a model. There have been proposed two methods based on this idea. One is ACM(Automatic Cognitive Modeling)(Langley & Ohlsson, 1984) and the other is HSMIS/THEMIS(Kono, Ikeda & Mizoguchi, 1993b). Both of the two methods act as domain-independent engine for student modeling.

2) ACM(Langley & Ohlsson, 1984)(Ohlsson & Langley, 1987)

ACM builds a model in terms of production rules corresponding to operators in the problem

space determined carefully, where applicability conditions of production rules are obtained in the learning process using a set of data. Typical primitive operators used for subtraction in ACM are as follows:

(1)***Add-Ten(number, row, column)***: This takes the ***number*** in a ***row*** and ***column*** and replaces it with that ***number*** plus ten.

(2)***Find-top(column)***: This takes a ***number*** from the top row of ***column*** and writes that ***number*** as the results for that ***column***.

And rules such as

Find-top:
If number-a is in column-a and row-a, and row-a is above row-b,
[and you are processing column-a],
[and there is no result for column-a],
[and you are focussed on column-a],
[and there is no number in column-a and row-b],
then write number-a as the result for column-a.

are obtained from a set of data, where conditions in the square brackets are put in advance. Note here that ACM is very different from LMS, though both represent knowledge in terms of rules. ACM tries to learn the conditions of the rules while LMS learns the sequence of the rule application. The resulting model corresponding to the buggy knowledge in ACM is represented in terms of rules with incorrect conditions with correct actions, while that in LMS in terms of rules with incorrect actions with correct conditions.

3) HSMIS/THEMIS(Mizoguchi, Ikeda, & Kakusho, 1988)(Mizoguchi & Ikeda 1990)(Kono, Ikeda & Mizoguchi, 1992)(Ikeda, Kono & Mizoguchi, 1993)(Kono, Ikeda & Mizoguchi, 1993a)(Kono, Ikeda & Mizoguchi, 1993b)

HSMIS proposed by the author's group synthesizes an SMDL(Student Model Description Language) program, which is a Prolog-based language taking four truth values such as true, unknown, false and fail, by observing both positive and negative examples which correspond to answers of the students to the problems given. HSMIS is based on MIS:Model Inference System developed by Shapiro(Shapiro, 1982) which synthesizes a Prolog program explaining the observed data. Roughly speaking, HSMIS is composed of SMIS and ATMS(Assumption-based Truth Maintenance System) developed by de Kleer(de Kleer, 1986). SMIS is an extended version of MIS for SMDL program. ATMS is employed to enable SMIS to cope with nonmonotonic behaviors of the students. THEMIS is the latest version of HSMIS augmented by adding MWC:Multi-World Controller to cope with the students with inconsistent knowledge in their heads. THEMIS builds models of such students by employing multi-world logic, since single world logic, which is a conventional logic, cannot build an inconsistent model. HSMIS/THEMIS is fully discussed in Chapter 15.

The search spaces of both ACM and HSMIS/THEMIS are usually very large, though they depend on the grain size of the primitives employed. The smaller the grain size becomes, the larger the search spaces become. Both can build models as precise as possible according to the grain size of the primitives used. Therefore, one has to pay close attention to how to use the models built and not to build unnecessarily precise models. Although modeling power of both of them is very high, the computational cost is expensive.

Although the two inductive methods share a lot of common characteristics, they differ in that HSMIS/THEMIS can cope with inconsistency and has sound theoretical foundation but ACM cannot. HSMIS/THEMIS has powerful and sophisticated mechanisms to cope with all the kinds of inconsistency appearing during the modeling process. And it is based on inductive inference theory of logic programming, which gives the method firm and sound theoretical foundation.

4.3 Other types of methods

Among several remaining modeling methods, I would like touch upon the following three methods:

(1) Process theory

Matz(Matz, 1982) proposed a process model. In her model, the student makes mistakes when applying knowledge to a new situation, in other words, he/she makes mistakes not by using incorrect knowledge but by misusing the correct knowledge. This idea remarkably characterizes her model, though it was not implemented. She classified the bugs into three categories two of which are described below.

1) Incorrect selection of interpolation procedure

When one comes across a new situation, he/she tries to apply his/her knowledge obtained in advance, in which he/she will have some difficulty of application of the knowledge, since it needs some interpolation procedure to fill the gap between the old and the new situations of the knowledge application. And he/she sometimes fails to interpolate the knowledge which causes errors. Typical mis-interpolation includes linear interpolation and generalization interpolation. The following are some examples of the former:

A(B+C) = AB + AC -> A(B*C) = AB * AC
(B+C)/A = B/A + C/A -> A/(B+C) = A/B + A/C

That of the latter is:

(X-3)(X-4) = 0
X-3 = 0 or X-4 = 0
X = 3 or X = 4
->
(X-A)(X-B) = K
X-A = K or X-B = K
X = K+A or X = K+B

2) The failure of assimilation of new knowledge

When acquiring a totally new concept, one sometimes fails to do it. Typical examples can be seen when kids first learn mathematics just after they learn arithmetics. For example, in arithmetics consecutive columns mean additive operation considering the position of columns, while it represents multiplication in algebra. Therefore, they sometimes give an answer X=8 to the following algebraic equation: 4X = 48. Another example is equality, "=". In arithmetics, the equality symbol means a command to calculate the formula on the left hand side, while it means a declarative relation stating both sides are equal to each other.

(2) Huang's method(Huang & McCalla, 1991)(Huang, 1993)

Huang has attacked the inconsistency problem. Except the author's group, he is the first researcher investigating modeling from inconsistent data. He proposes logic of attention which has

two kinds of memories such as short-term and long-term memories. The system tries to maintain the consistency only of the data in the short-term memory. It does not care about the consistency in the long-term memory. His method mainly concerns with the attention shifting operations. Although it is theoretically deep and sound, the method is based not on predicate logic but on propositional calculus. Therefore, the applicability of the method is limited.

(3) Constraint-based approach(Ohlsson, 1993)

Considering that functions the student model should perform and that intractability of modeling, Ohlsson proposes a new approach to student modeling called constraint approach in which domain knowledge is represented as sets of constraints and student model is represented as constraint violation. The idea seems good, since it can avoid many of the difficulties of the current modeling technology is suffering from. However, one has to identify sets of constraints which covers all the information the tutoring module requires which is not an easy task.

4.4 Discussions

We have thus far discussed various student modeling techniques from computational point of views. Although many modeling techniques have been proposed to date, completely satisfactory methods are not found yet. But I think the technology grows up slowly but steady. Table 1 summarizes the characteristics of the above methods.

Table 1 Evaluation of student modeling methods.

System name	Analytic/ inductive	Mal-func-1	Mal-func-2	Mal-struct.	Inconsist-ency	Theoretical foundation	Question auto.
Overlay	analytic	yes	no	no	no	low	NA
(Hoppe, 1994)	analytic	yes	yes	no	no	middle	NA
(Self, 1993a)	analytic	yes	yes	no	no	high	yes
IDEBUGGY	inductive	yes	yes	yes	partly	low	yes
Perturbation	inductive	yes	yes	yes	no	low	no
ACM/DPF	inductive	yes	yes	yes	partly	middle	no
THEMIS	*inductive*	*yes*	*yes*	*yes*	*yes*	*high*	*yes*

5. Basic Techniques of HSMIS/THEMIS

I explain the key technology employed in HSMIS/THEMIS in this section.

5.1 Building blocks of the framework

In order to obtain theoretically sound mechanisms, they should be designed at least independently of teaching materials. Mizoguchi and Ikeda have investigated a guide line for designing modules including student modeling module(Mizoguchi & Ikeda, 1990). They call a module designed according to the guide line a building block which is composed of a problem space, a general problem solver and heuristics. A building block can solve any problem defined in the problem space, where definition of the space is very important to realize generality. Heuristics specific to the domain, if available, enables it to solve the given problem more efficiently. Thus design process of building blocks consists of the following three steps:

(1) Formal description of the problem space,
(2) Implementation of a general problem solver for that space, and
(3) Definition of heuristic knowledge available.

5.2 Inductive inference

The inference to obtain general rules which explains a set of data is called inductive inference. For example, one can easily induce a rule "Humans are mortal" by observing the following data:

John was dead.	John is a human.
Mary was dead.	Mary is a human.
Tom was dead.	Tom is a human.

Description of this in terms of Prolog is shown as follows:

```
mortal(john).  human(john).
mortal(mary).  human(mary).
mortal(tom).   human(tom).
->
mortal(X) :- human(X).
```

Another example in Prolog looks like the following:

```
append([a], [b], [a,b]), true.    append([a], [], [a]), true.
append([c], [b], [b,c]), false.   append([a], [b,c], [a,b,c]), true.
->
append([], [X], [X]).
append([H|X], [Y], [H|Z]) :- append(X, Y, Z).
```

The predicate "append(X,Y,Z)" means concatenation of the two lists X and Y obtains a list Z. The two clauses constitutes a Prolog program and covers the above four facts. In this sense, to synthesize such a Prolog program is a kind of inductive inference. Let us translate this procedure into our student modeling problem. Suppose a simple chemical reaction is employed as a domain in which facts and rules are written in Prolog as follows:

```
acid(cl,1).
acid(so(4),2).
base(ca,2).
base(ba,2).
precipitation(ca,so(4)).
precipitation(ba,so(4)).
salt(X,Y):-base(X,_), acid(Y,_).
react(X,Y,Z):-salt(X,Y), precipitation(X,Z).
```

The first fact means "Cl" is an acid, the third "Ca" is a base, and the fifth CaSO(4) precipitates, respectively. Furthermore, the first and second clauses(rules) mean that if X is a base and Y is an acid then XY is a salt and if XY is salt and XZ precipitates then reaction XY + HZ -> XZ + HY occurs, where H denotes a hydrogen atom, respectively. For example, when X = Ca, Y = SO(4), and Z = Cl, it means the reaction

CaSO(4) + 2HCl -> CaCl(2) + H(2)SO(4)

occurs, since CaSO(4) is a salt and CaCl(2) precipitates.

Then what the inductive inference system has to do is to infer a set of clauses (program) from the data as follows:

react(ca, so(4), cl(2)), true
react(ba, co(3), cl(2)), false
.....
precipitation(ca, cl(2)), true
.....

In our situation, this information is easily obtained by giving problems to the students and observe their answers as shown below.

CaSO(4) + 2HCl -> CaCl(2) + H(2)SO(4) ? yes.
BaCO(3) + 2HCl -> BaCl(2) + H(2)CO(3)? no.
.....
Does CaCl(2) precipitate? yes.
.....

where the student's answers are underlined . If a model like

react(X,Y,Z):-salt(X,Y).

is obtained, then it shows the student forgets the second condition, that is, precipitation and, if we obtain a model

react(X,Y,Z):-salt(X,Y), precipitation(X,Y).

then it shows the student mis-understands the reaction occurs in a reverse direction, which satisfactorily models his/her mis-understanding of the chemical reaction.

The above discussion shows Prolog-based inductive inference can be used as a student modeling technique. The basic mechanism used in HSMIS/THEMIS is Shapiro's MIS(Model Inference System) which is summarized in the following:

5.3 MIS(Shapiro, 1982)

MIS is based on the generate & test paradigm and is composed of major two modules such as refinement operators(generator) and PDS:Program Diagnosis System(tester). The former generates a candidate of Prolog clauses and the latter finds bugs if the current model(a set of clauses) does not explain all the data given thus far. When it finds a model explaining all the data, MIS outputs it as a model of the student. PDS identifies two types of bugs such as incorrect clause bug and uncovered goal bug. The former is a case where a positive goal is not covered by the model and the latter is found in a case where the model derives a positive answer which should be negative. In other words, the model answers "no" to the question(problem) that the student answered "yes" and it answers "yes" to that he/she answered "no". In the former case, in which the model is said to be too weak, it lacks a clause to cover the goal and in the latter case, in which it is said to be too strong, it contains at least one clause which is responsible for the incorrect answer.

Roughly speaking, when the model generates incorrect answers, MIS revises it in two ways according to the characteristic of the incorrectness. Note here that in the modeling phase, "a *correct* answer of the model" does not mean the same answer as that made by the domain knowledge but the same one as that made by the student, since the model should be evaluated its correctness against how well it represents answers of the students.

PDS finds a bug in the current model by tracing the computation tree during which it asks the student truth values of necessary facts. The questions are made by translating its behavior in student modeling situations, that is, by giving problems to the student and observing the answers to them. Suppose MIS needs to know the truth values of the predicate "precipitation(ca, so(4)). Then, the problem generated by MIS looks like

precipitation(ca,so(4))?

which is translated by the interface as

Does CaSO(4) precipitate?

Basically the refinement operator is a set of operators which generates all the forms of clauses

satisfying given conditions which restricts the search space. It requires a set of predicates necessary for composing the target clauses for each of them. For example, to synthesize the clause react(X,Y,Z) it requires salt(_,_), precipitate(_,_), and some unnecessary predicates the student may use. Then, what the refinement operator does is to generate possible clauses by choosing and combining appropriate predicates from the set of predicates given and by adjusting the structure of arguments of each predicates. Thus, MIS can model students without knowing domain-specific bugs.

The algorithm of MIS is summarized as follows:

```
Let T = {nil}
Repeat
    Read the next fact.
    Repeat
        While "the model T is too strong" do
            Identify a buggy clause using PDS and delete it from T.
        While "the model T is too weak" do
            Identify an uncovered goal using PDS and add a clause to T using the
            refinement operators.
    Until T is neither too strong nor too weak.
    Output T as a solution.
Until there is no fact to read.
```

SMIS is an extended version of MIS in that it synthesizes not a Prolog but an SMDL program which takes four kinds of truth values as mentioned above. The difference between MIS and SMIS comes from those between the truth values. The strength of the four truth values determined as

true > fail > unknown > false,

from which strength and weakness of a model are defined, which in turn defines behaviors of PDS. The details are omitted here because it is too technical.

5.4 Coping with inconsistency

As is described in the above, HSMIS/THEMIS copes with inconsistent data which had been one of the serious problems in student modeling. The author carefully classified inconsistency appearing in the student modeling process and devised powerful methods dealing with inconsistency in the framework of inductive inference. HSMIS employs ATMS to manage the inference process of SMIS and to provide SMIS with consistent data in order to enable it to induce a model from consistent data. In other words, HSMIS works to avoid inconsistency, since many types of inconsistency is unnecessary information.

Note, however, that there is a kind of inconsistency which should not be avoided, rather we have to deal with it as it is. Those are the cases where the students have inconsistent knowledge in their heads. In such cases, we have to model their inconsistent knowledge as it is, and utilize the inconsistency to tutoring, since effective tutoring such as Socratic method can be realized using the inconsistency. THEMIS is designed and implemented to satisfy this requirement by extending HSMIS. So, THEMIS can avoid unnecessary inconsistency of the data and can model necessary inconsistency. HSMIS/THEMIS is implemented in CESP, object-oriented Prolog developed by ICOT, the fifth generation computer project. Chapter 15 discusses HSMIS/THEMIS in depth.

6. Concluding Remarks

We have discussed several issues in student modeling research and overviewed some typical modeling methods. Although a lot of work has been done, a number of problems still remain unsolved.

Much more theoretical research is required in order to make the research clearer and sophisticated. Although it is difficult to build a satisfactory student model, I do not think it is intractable. Rather, student modeling is a challenging and attractive research issue. In addition to the theoretical research, robust modeling techniques have to be devised. Furthermore, efficient tutoring strategies less dependent on the quality of the student model are also valuable.

In general, we have to know more about cognitive process of human problem solving. There exist various kinds of difficulties in human problem solving. Nevertheless, some are left unexplored. Deep consideration on cognitive model will help us design a more effective and efficient modeling method.

Acknowledgement: The author is grateful to Dr. Mitsuru Ikeda for his valuable comments.

<REFERENCES>

Beller, S. & Hoppe, H. U. (1993). Deductive error reconstruction and classification in a logic programming framework, Proc. of the AI-ED93, Edinburgh, 433-440.

Brown, J.S. & Burton, R. R. (1978). Diagnostic models for procedural bugs in basic mathematical skills, Cognitive Science, Vol. 2, No. 2, 155-191.

Burton, R. R. & Brown, J. S. (1979). An investigation of computer coaching for informal learning activities, International Journal of Man-Machine Studies, Vol. 11, No. 11, 5-24.

Burton, R. R. (1982). diagnosing bugs in a simple procedural skill, in D. H. Sleeman et al. (Eds.), Intelligent Tutoring Systems, Academic Press, London, 157-183.

Carr, B. & Goldstein, I. P. (1977). Overlays. A theory of modeling for computer-aided instruction, AI Lab Meno 406, MIT, Cambridge, Massachusetts.

de Kleer, J. (1986) .An assumption-based TMS, Artificial Intelligence, Vol.28, No.2, 127-162.

de Kleer, J. & Williams, B. (1987). Diagnosing multiple faults, Artificial Intelligence, Vol.32, 97-130.

Dillenbourg, P & Self, J. (1992). A framework for learner modelling, Interactive Learning Environments, 2, 2, 111-137.

Hoppe, H. U. (1992). Deductive error diagnosis in intelligent tutoring systems, Proc. of the first East-West Conference on Emerging Computer Technologies in Education, Moscow, 147-152.

Huang, X. & McCalla, G. (1991). Using attention in belief revision, Proc. of AAAI-91, 275-280.

Huang, X. (1993). Inconsistent belief, attention, and student modeling, J. of AI in Education, Vol.3, No.4, 417-428.

Ikeda, M., Kono, Y., & Mizoguchi, R. (1993). Nonmonotonic model inference - A formalization of student modeling-, Proc. of IJCAI'93, Chambery, France, 467-473.

Kono, Y., Ikeda, M., & Mizoguchi, R. (1992). To contradict is human - Student modeling in consistency, Proc. of ITS-92, Montreal, 451-458.

Kono, Y., Ikeda, M., & Mizoguchi, R. (1993a). A modeling method for students with contradictions, Proc. of AI-ED93, Edinburgh, 481-488.

Kono, Y., Ikeda, M., & Mizoguchi, R. (1993b). THEMIS: A nonmonotonic inductive student modeling system, AI Technical Report, ISIR, Osaka University, AI-TR-93-3.

Langley, P. & Ohlsson S. (1984). Automated cognitive modeling", Proc. of National Conference, on Artificial Intelligence, Austin, Texas, 193-197.

Matz, M. (1982).Toward a process model for high school algebra, in D.H. Sleeman et al. (Eds.), Intelligent Tutoring Systems, Academic Press, London, 25-50.

Mizoguchi, R. & Ikeda, M. & Kakusho, O., (1988). An innovative framework for intelligent tutoring systems in P. Ercoli and R. Lewis eds., Artificial Intelligence Tools in Education, North Holland,105-120.

Mizoguchi, R. & Ikeda, M. (1990). A generic framework for ITS and its evaluation, Proc. of International Conference on ARCE, Tokyo, 303-312.

Ohlsson, S. & Langley, P. (1987). Identifying solution paths in cognitive diagnosis, in Mandl, H. et al.(Eds) Learning Issues for Intelligent Tutoring Systems, Springer-Verlag, New York.

Ohlsson, S. (1993). Constraint-based student modeling, J. of AI in Education, Vol.3, No.4, 429-447.

Otsuki, S. & Takeuchi, A. (1985). Intelligent CAI system based on teaching strategy and learner model, Proc. of WCCE 85, 463-468.

Self, J. (1988). "Bypassing the intractable problem of student modeling", Proc. of ITS' 88, Montreal, 18-24.

Self, J. (1992). Model-based cognitive diagnosis, J. of User Modeling and User-Adapted Interaction, Vol.3, 89-106.

Shapiro, Y. (1982). Algorithmic program debugging, MIT Press.

Sleeman, D.H. (1983). Inferring student models for intelligent computer-aided instruction, In Michalski, R.S. et al. Eds. Machine Learning. An Artificial Intelligence Approach, Morgan Kaufmann.

Takeuchi, A. & Otsuki, S. (1987). Formation of learner model by perturbation method and teaching knowledge, Transactions of Information Processing Society of Japan, Vol. 1, No. 1, 54-63(in Japanese).

Wenger, E. (1987). Artificial Intelligence and Tutoring Systems - Computational and cognitive approaches to the communication of knowledge -, Morgan Kaufmann Publishers, Inc.