Nonmonotonic Student Model Inference System(Draft)

Riichiro Mizoguchi, Mitsuru Ikeda, and Yasuyuki Kono

I.S.I.R., Osaka University
8-1, Mihogaoka, Ibaraki, 567 Japan
Email: miz@ei.sanken.osaka-u.ac.jp

1. INTRODUCTION

The authors have been involved in investigation of ITS for several years[Kawai, 1985] [Mizoguchi, 1988][Ikeda, 1988a][Ikeda, 1988b][Ikeda, 1989][Mizoguchi, 1990][Kono, 1992] [Kono, 1993a][Kono, 1993b][Ikeda, 1993a][Ikeda, 1993b]. This paper reviews their research activities on student modelling for ITS and discusses the major results obtained thus far. The next section presents the underlying philosophy and an outline of a framework for ITS called FITS in which our student modelling module plays a crucial role. Section 3 discusses the design philosophies of the student modelling methods THEMIS. The rests of the paper present conceptual level description of THEMIS. Readers interested in the technical details are advised to refer to [Ikeda, 1993a][Kono, 1993].+

+ This paper neither mentions other methods nor compares the proposed methods to related work in order to avoid duplication with the accompanying tutorial paper[Mizoguchi, 1993].

---

2. OVERVIEW OF THE RESEARCH

2.1 Research objectives

Our research has been conducted aiming at the following two major goals:
1) to reveal inherent computational structure of ITS, and
2) to provide powerful artificial intelligence techniques with ITS community.

As a result, much attention has been paid to designing a domain-independent framework of ITS. Most of the existing ITSs are domain-dependent in which architecture and the knowledge embedded is deeply related to respective domain knowledge. Therefore, it is not clear which knowledge and which part of the system make essential contributions to the success of the system.

How much do we know about the common architecture of ITS? What architecture, what knowledge and what mechanisms can be reused for building another ITS in different domains? The authors have been investigated these issues for several years. FITS, Framework for ITS, is designed based on the results of the investigation. FITS represents inherent problem solving structure of tutoring independently of each teaching material, which makes it easy to build an ITS, since control structures common to most ITSs are already embedded in the framework. One of the contributions FITS makes is not only to show an applicability of advanced artificial intelligence techniques but also to demonstrate that ITS research gives us a strong motivation to devise new sophisticated techniques especially for student modelling.

2.2 The framework

An ITS is composed of the following three major functional modules:
(1) Expertise module,
(2) Student model module, and
(3) Tutoring strategy module.

These are further divided into a few primitive tasks. FITS is composed of seven primitive modules such as student model interpretation, SMDL interpreter, THEMIS, Bug identification, Bug causality analysis, Expertise, Tutoring, and Scheduling modules each of which represents inherent task required in ITS. Among these modules, the first four collectively constitute a student model module which plays an essential role in FITS, since performance of an ITS depends largely on how well it knows the student. We developed a modelling language called SMDL(Student Model Description Language) based on Prolog [Mizoguchi, 1988]. In our framework, the student model is represented as an SMDL program, so modelling students can be considered as a program synthesis problem and instruction can be considered as debugging of the SMDL program. Therefore, tutoring is formulated as a two-step procedure composed of program synthesis and its debugging as shown below, which is a skeletal idea of our framework.

Intelligent tutoring = Program synthesis + Debugging.

Now a serious question arises: How can we synthesize an SMDL program? An inductive inference system called SMIS(Student Model Inference System) is also developed as an answer to this question and it works as a central engine of the framework. In the current implementation, the expertise module consists of only one building block, that is, Prolog interpreter which interprets the teaching material described in Prolog. FITS also has an efficient scheduler which makes an appropriate decision on what to do next. It is designed based on SOAR architecture[Laird, 1986] to realize highly flexible scheduling.

FITS also has a set of domain-independent tutoring strategies. One may think tutoring cannot be domain-independent, since it seems to require a lot of domain-dependent data or facts. It is true that tutoring strategies use domain-specific data and facts in explanation and other interactions. By domain-independent tutoring strategy, we mean their mechanisms are domain-independent. Furthermore, when and what strategies the system should take is also determined as domain-independently as possible. The strategies consist of two major groups such as one based on explanation and the other based on hints using domain-specific examples(data). For detailed description on tutoring strategy and scheduling, refer to[Mizoguchi, 1990][Ikeda, 1993b]. FITS can easily incorporate domain-dependent knowledge into its mechanism which is its another advantage.

2.3. Building blocks of the framework

Since ITS can be viewed as an expert system in education, expert system technologies can be applied to construction of ITSs. Most of the conventional expert systems are constructed using production rules which provide fairly low-level description of expertise. Systems based on production rules do not reflect inherent structure of the task, so one may have a lot of difficulty in describing expertise in terms of rules. Recently, the concept of generic tasks [Chandrasekaran, 1986] attracts much attention which can be building blocks of expert systems. By generic tasks, we mean domain-independent but task-dependent chunk of control structures. In order to obtain a generic framework, all the modules have to be designed independently of a teaching material. We investigated the primitive modules to identify domain-independent structures which act as building blocks of ITS.

Each building block is designed as a general problem solver for its corresponding generic task. Implementation of building blocks is based on a problem space approach. It consists 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.

When formulation of the problem space is made in an adequate level of abstraction, high generality and efficiency can be attained simultaneously. Moreover, users of the framework can encode the knowledge in terms of vocabulary at an appropriate conceptual level.

3. DESIGN PHILOSOPHY OF THEMIS

Discussions on student modelling are made from two major view points: Computational and cognitive ones. There is no need to stress the importance of cognitive aspects of student modelling. Recently, on the other hand, researchers involved in AI in education field tends not to appreciate the importance of the computational aspect. ITSs realized on computers largely depends on computational technology available, since developers of ITSs cannot design any system without knowing what he/she can do with computers. Well-balanced considerations on computational and cognitive issues are crucial to steady and healthy progress of the research. For this reason, the authors have taken the computational standpoint in the research on ITS. They believe technological progress provides the AI in education researchers with powerful tools for designing and building educational systems.

Modelling human understanding is a really challenging and ambitious problem. Although we know much of the research results on model inference and learning done in AI research, really important and exciting tasks to do are not only to inherit them but also to augment, improve, and invent theories and techniques tailored for ITSs.

There are several dimensions for characterizing student modelling methods:

1) Inductive vs. analytic
2) Mal-function vs. mal-structure
3) Theoretical foundation
4) Inconsistency
5) Representation primitives
6) Multiple observations vs. single observation
7) Link to how to use the model
8) Accuracy-cost trade-off
9) Automated question generation
10) Executability

This section discusses the design philosophies of THEMIS with respect to these ten issues.

1) Inductive vs. analytic
Modelling is viewed as an inductive process in which a representation explaining observed data is built. Although a student model does not have to explain all the behavior of the student, inductive modelling methods usually build more comprehensive models than analytic methods. Furthermore, inductive modelling of the students is one of the challenging topics. For these reasons, inductive approach is taken.

2) Mal-function vs. mal-structure
Mal-structure is modelled in THEMIS. Obviously, modelling methods covering mal-structure of the student knowledge is more powerful than those covering only mal-function. Mal-structure can be modelled only by using inductive approach.

3) Theoretical foundation
Domain-independent and theoretical foundation for the student modelling mechanism should be explored. It contributes to both clarification of the inherent property of student modelling problems and to articulation of the scalability and reusability of the proposed mechanisms. Efficient domain-specific modelling methods are of course important. However, we do need general methodologies and theories of student modelling. In order to develop such a method supported by a firm theoretical background, model inference in the logic paradigm is employed in THEMIS.

4) Inconsistency
Viewing student modelling as an inductive inference from observed data, data are student's answers to the problems given to them and the model is one explains the student's behaviors observed. In ITS, however, there is no guarantee that the student responses are always consistent, since he/she changes his/her belief, sometimes makes careless mistakes, and may have inconsistent knowledge in his/her head. Therefore, the inductive inference in student modelling is essentially non-monotonic. The modelling system is required to maintain the consistency of the set student's responses. THEMIS has powerful mechanisms to cope with the inconsistency.

5) Representation primitives
In order to build a model, we need a set of primitives in terms of which the model is described. Roughly speaking, there are two approaches to this problem, that is, one based on bugs obtained by analyzing the students answers of the domain and the other based on small primitives of the domain knowledge. While the former requires a lot of work to enumerate and analyze bugs but modelling is generally efficient, the latter is free from bug analysis and has high generality but modelling is often inefficient due to the large search space. THEMIS takes the latter approach in order to pursue the generality employing some techniques to narrow down the search space.

6) 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 to obtain reliable models and to realize quick and timely didactic feedback. However, modelling from one observation and quick didactic feedback can realize neither comprehensive nor well-organized tutoring. THEMIS employs modelling from multiple observations to realize the sophisticated and comprehensive tutoring.

7) Link to how to use the model
Tutoring behavior depends largely on student model. If one has a poor model, he/she cannot control the tutoring behavior well. This motivates many researchers to design methods for building student models of high fidelity, which causes people to say the student modelling problem is "intractable". Therefore, designers of student modelling methods have to be careful not to build overspecified methods. That is, a student model built should provide necessary and sufficient information with a tutoring module which realizes adaptive tutoring. In other words, a student model cannot be designed independently of tutoring module's requirements. The tutoring module in FITS is based on sophisticated hints using examples which make the students notice their incorrectness such as missing and unnecessary factors and Socratic method which tries to make the student notice contradictions within him/her. In order to operate such tutoring methods, we have to precisely identify the student's understanding state which gives THEMIS a justification of the grain size of its model description.

8) Accuracy-cost trade-off
Building precise models with high fidelity is usually computationally expensive. So, designers have to solve another trade-off between accuracy and computational cost. THEMIS builds a model of so fine grain size that it is computationally expensive. But it runs efficiently enough on the current EWSs by introducing several speeding up techniques in it.

9) Automated question generation
Some of the student modelling methods proposed to date do not ask the students questions during the course of modelling, which causes the problem unnecessarily difficult. When the system encounters ambiguity, it is desirable to ask the student to give necessary information by asking him/her appropriate questions. Student modelling is different from learning in that it can ask reasonable amount of questions, since asking students questions is equivalent to giving them problems to solve in tutoring situation. THEMIS has an automatic question generation mechanism to make the model built as reliable as possible.

10) Executability
When student models are executable, the system can predict the answers of the student and make problems appropriate for him/her. Thus executable models enhance the performance of the tutoring. Especially, the tutoring methods mentioned above requires an executable model. THEMIS employs a logic-based executable language for describing the student model.

The following sections present the conceptual level description of THEMIS.

4. SMDL: STUDENT MODEL DESCRIPTION LANGUAGE

In order to enable theoretical discussion on the modelling methods, logic is employed in THEMIS. It provides us with firm and theoretical foundation. For the description language of a student model, however, predicate logic is not powerful enough for the purpose in that it only takes two kinds of truth values. First of all, the modelling language has to take two truth values representing the student positive answer, e.g., "Rice grows in Tokyo", and negative answer, e.g., "Rice does not grow in Kiev". It is natural to use "true" and "false" for representing these answers. Furthermore, considering that a student model is a system's belief about the student knowledge state, the modelling language has to take two additional truth values representing if the system believes the current model explains an observed data correctly or not. When we interpret "true" and "false" as the system believes the model is correct, we need one more value representing it does not believe the model is correct. We denote the third value as "fail" which means the system fails to explain an observed data by the current model. Basically, a language taking these three kinds of truth values suffices for many cases. But we sometimes need to represent the student does not know the answer, i.e., answers like "I do not know if rice grows in Osaka or not". This suggests the fourth value "unknown".

SMDL is a student modelling language which is executable and can simulate the problem solving behavior of the student[Mizoguchi, 1988]. It is an extended version of Prolog and takes the above four truth values such as "true", "false", "unknown(unk)" and "fail". The first three values correspond to "success" in Prolog and the last to "fail". "true" stands for the student answers "yes", "unknown" for "unk", "false" for "no" and "fail" for the system does not know what answer the student will return. Facts are represented in SMDL as follows:

```
    temperate(japan, true).
    torrid(japan, false).
    fertile(japan, unk).
```

These facts represent the student's knowledge:

    "I know Japan is not torrid but temperate, but do not know
    whether it is fertile or not."

Clauses(rules) in SMDL are also similar to those in Prolog except they have an additional argument for truth values introduced above. In SMDL, the truth value of a goal is determined by applying the OR operator shown in ??Table 1 (b) to the truth values derived from all the clauses whose heads match with the goal. The truth value derived from each clause is defined by applying the AND operator shown in ??Table 1 (a) to the values of all the predicates appearing in its body. This evaluation is performed by SMDL interpreter implicitly. Some examples are shown below.

```
    grow(Plant, Place, T1)::-
        torrid(Place, T2), wet(Place, T3).
    grow(Plant, Place, T4)::-
        temperate(Place, T5).
```

Given a goal grow(tree, japan, T), SMDL interpreter evaluates the subgoals torrid(japan, T2), wet(japan, T3) and temperate(japan, T5) in this order and obtains a truth value T according to the following manner:

    T = T1 OR T4 = (T2 AND T3) OR T5.

```
\begin{table*}\begin{center}
    \caption{Definition of \And \  and \Or . }
     \label{tab:Truth table}
\parbox[t]{6cm}{\small
\hspace*{1.5cm}  (a) \And \ operator \\
\begin{tabular}{l| |l|l|l|l|}
$\wedge$       & true  & unk.  & false & fail  \\ \hline\hline
true         & true  & unk.  & false & fail  \\ \hline
unk.         & unk.  & unk.  & false & fail  \\ \hline
false         & false & false & false & fail  \\ \hline
fail         & fail  & fail  & fail  & fail  \\ \hline
\end{tabular}

}
\parbox[t]{6cm}{\small
\hspace*{1.5cm}(b) \Or \ operator\\
\begin{tabular}{l| |l|l|l|l|}
$\vee$       & true  & unk.  & false & fail  \\ \hline\hline
true        & true  & true  & true  & true  \\ \hline
unk.        & true  & unk.  & unk.  & fail  \\ \hline
false       & true  & unk.  & false & fail  \\ \hline
fail        & true  & fail  & fail  & fail  \\ \hline
\end{tabular}
}
\end{center}
\end{table*}
```

5. SMIS: STUDENT MODEL INFERENCE SYSTEMS

SMIS is an inductive inference system for SMDL. It is an extended version of MIS[Shapiro, 1982]. A pair of a problem and an answer to it is called an oracle and is used for inductive inference as data to be covered by the model built. Given an initial model, which is usually the correct knowledge, SMIS builds a model(an SMDL program) by applying the following two operations repeatedly to the model:

(1) removal of an incorrect clause and

(2) addition of a new clause

until the model comes to be able to cover all the oracles given. An incorrect clause which has a logical refutation by oracles is identified by SMDS(Student Model Diagnosis System). SMDS traces computation process of the current model and checks the results with student's answers. An uncovered oracle to be covered by the model is also identified by SMDS. To cover it, SMIS searches for a new clause to insert into the current model. The new clause must have a logical support by oracles. Suppose that we have an uncovered oracle and a clause whose head and body match with some oracles. Then the clause is added to the current model. Candidates of the clause to be added are generated by refinement operators, which are defined by modifying those defined in MIS. Refinement graph is a directed graph whose nodes are clauses of SMDL and whose arcs correspond to refinement operations. The child nodes of a node are produced by applying refinement operators to the node. If a node is not supported logically, its child nodes are not either. Using this property, SMIS can prune search space for new clauses in the refinement graph. Generality of the student modelling method with SMIS is sufficiently high, since it can build any models which can be described in terms of SMDL.

6. HSMIS: HYPOTHETICAL SMIS

The inductive inference algorithm described above is based on the assumption that all the oracles are consistent, that is, student's answer are consistent. Unfortunately, however, the assumption does not always hold. Students change their minds and sometimes make careless mistakes. Therefore, SMIS must cope with inconsistent oracles. This requires nonmonotonic inductive inference. In our modelling method, ATMS: Assumption-based Truth Maintenance System[de Kleer, 1986] is employed for this purpose. SMIS augmented with ATMS is referred to as HSMIS: Hypothetical SMIS.

6.1 Inconsistency

Inconsistency appearing in the student modelling process is classified into the following four categories according to their causes:

1) Oracle contradiction caused by change of students' understanding:
Students' learning process is essentially attained by acquiring new knowledge which necessarily causes change of their understanding. The consistency of their answers within the whole learning process can be easily lost, since they believes based on their current knowledge independently of their previous knowledge.

2) Oracle contradictions caused by slips:
Students often make careless mistakes. The set of oracles that contains slips is inconsistent.

3) Student knowledge contradictions:
Students sometimes have inconsistent knowledge in their heads which also causes contradictory oracles.

4) Assumption contradiction in modelling:
Inductive inference is essentially a hypothetical process, since the correctness of inferred model is not guaranteed. The expectation of student's answer deduced from the current student model is often different from new oracles, when the current model does not completely represent his/her current status. Assumptions which were assumed when the current model was inferred become inconsistent with the set of oracles.

The causes of the first three are related to students and referred to as student contradictions, while that of the last is related to inductive inference itself and is referred to as modelling contradictions. On the other hand, the third inconsistency (contradiction) is different from the other three in how to be dealt with. The other three contradictions are unnecessary ones, while the third should be modelled as it is, since the contradictions in the students' heads provide tutoring modules with valuable information. We call the former type of contradictions single world contradictions, since the student should be modelled in a single consistent world. And we call the latter multi-world contradictions, since the students with such contradictions often be modelled as possessing separate conflicting worlds each of which is consistent. HSMIS deals with the former inconsistencies and THEMIS deals with the latter.

6.2 ATMS

This subsection briefly explains about ATMS which works cooperatively with a problem solver(inference system) to manage the inference process and the data inferred by the problem solver. The information given by inference system, SMIS in HSMIS case, takes the form of N1, N2, ..., Nk => D which means the data D is derived from a set of the data N1, N2, ..., Nk which is called a justification of D. The data dealt with in the inference system are classified into three kinds of data, i.e. premise data, assumed data and derived data. The premise is defined as a datum that is true under any context. The assumed data are the ones produced with an assumption that they are held without depending on other data. The derived data are the ones inferred from other data. In HSMIS, assumptions and derived data correspond to oracles and clauses in the model, respectively. Traces of the supporting reasons finally reach the premises or the assumptions when the justification is pursued from the derived data. That is to say, a set of assumptions that the individual derived data depend on can be calculated. A set of assumptions is referred to as an environment. It is one of the major tasks for ATMS to record justifications informed from the inference system and calculate a consistent environment where the data can be inferred. When derivation of the contradiction, which is usually defined by a set of rules in the inference system, is informed, ATMS calculates the nogood environment which is a cause of the contradiction and is recorded in a nogood record. The environment included in the nogood record can be regarded as the incorrect combination of the assumptions. ATMS maintains the consistency in the inference process by using the nogood record. The inference system selects a new consistent environment, which does not include the nogood record elements, and continues inference.

A situation in the problem solving process is called a context, which is defined with a set of data held in the situation. An environment deriving all the data included in the context is called a characteristic environment of the context. When a contradiction is derived, the inference system ceases to solve the problem in the contradicted context and transfer to a new consistent characteristic environment. With regard to the nodes that have been derived until that time, ATMS determines whether the nodes hold (in) or do not hold (out) in the new characteristic environment. Thus a new context is composed of a set of "in" nodes.

6.2 Overview

HSMIS consists of SMIS, ATMS, Virtual oracle generator(explained in 6.3.2), and Conflict resolving system(CRS). The main task of ATMS is to manage consistency of a set of assumptions used by the problem solver, SMIS. In HSMIS, assumptions are oracles, since every activity in HSMIS is ultimately dependent on oracles and inconsistency appears among oracles. Virtual oracle generator is responsible for decreasing the questions made by the model inference system by generating virtual student answers based on the reliability of the student without asking the student questions. CRS resolves the inconsistency identified by changing the environment. Inconsistencies(contradictions) in HSMIS is defined as rules according to the causes described above[Kono, 1993b].

The control flow of HSMIS is as follows:
1) Given student answers, oracle generator generates virtual oracles if necessary
and pass them to ATMS together with the "real" oracles.
2) SMIS informs ATMS of all the inference process. When a contradiction is
informed, ATMS computes the environment responsible for the inconsistency based on the information given up to that point and store it in the nogood record.
3) SMIS asks CRS to resolve the inconsistency.
4) According to the causes of inconsistency identified, CRS selects an appropriate
set of consistent oracles by asking ATMS to check their consistency.
5) ATMS answers the queries by inspecting the nogood record and
6) Passes the control to SMIS together with the restored data when the new
environment(a set of oracles) is consistent.

6.3 Controlling the modelling process

HSMIS tries to model the student from his/her behaviors during which it automatically asks questions which contributes to identification of inappropriate clauses and disambiguation of alternative model selection. In other words, HSMIS asks questions regardless of their appropriateness in the sense of tutoring. Improvement of this requires some control mechanism of the HSMIS behavior. In order to make the inference efficient, it has to employ some heuristics to keep the search space in a reasonable size. Furthermore, HSMIS has to efficiently cope with inconsistency caused by various causes. This subsection describes several additional mechanisms introduced to augment the HSMIS.

6.3.1 Heuristics for student modelling

A refinement graph spanned by refinement operators defines the search space of each clause of interest. However, it originally does not have any a priori knowledge of bugs. So, it always tries to find out a clause from a fixed root clause independently of domain knowledge. Note here that we can introduce the concept of bug when we know the domain knowledge. Given some typical bugs specific to the domain knowledge under consideration, HSMIS searches for clauses starting from these bugs, which often makes the search very efficient. When it fails, it begins to search from the original root. Therefore, introduction of heuristics does not loose any generality. Note here that this is a very important characteristics. HSMIS works without any heuristics and it is easy for developers to introduce heuristic knowledge about the domain-specific bugs into HSMIS to enhance the modelling performance.

6.3.2 Virtual oracles

Let us discuss the initial model problem. There are two alternative initial models: one is empty which means the teacher does not know about the student in advance and the other is complete model(teaching material itself) which means the teacher assumes students are usually understand the material very well. Although the former case seems reasonable, the system tends to ask many questions to get a lot of information of how well the student understands the domain knowledge. On the other hand, the latter case does not require many questions at least for excellent students. This characteristics is very reasonable in tutoring. Therefore, we decided to employ the latter. However, it still remains a problem. HSMIS has to have a justification for everything in the model, even if it is the initial model. One cannot simply put the correct model into the student model in HSMIS without any justification. In order to cope with this problem, we devised "virtual oracles" which serve as justifications for the initial model or models with uncertain justifications. When the system tries to put a clause into the current model without "real" support, for example, Virtual oracle generator generates virtual oracles which support the clause. HSMIS is already designed to cope with inconsistency of oracles, the virtual oracles can be dealt with by HSMIS very easily. Needless to say, when contradiction occurred, the clauses supported by virtual oracles are the first candidate to withdraw.

When the teacher believes his/her student is wise enough, he/she asks less questions by replacing the necessary information with assumed answers expected by the domain knowledge. When the assumed and expected answer turns out to be no longer true after the inference proceeds, HSMIS withdraws the model supported by the oracles and back up to the point which causes the problem. This is another example of "virtual oracles". HSMIS distinguishes between "real" and "virtual" oracles by labeling them and manage them with the aid of ATMS. This mechanism helps decrease the questions given from the HSMIS.

6.3.3 Meta-oracles

HSMIS accepts as oracles not only facts but also a clause itself. Students sometimes want to say his/her knowledge in the form of rules instead of facts. And the system sometimes wants to ask the student the reason why he/she answers a question that way. The following is an example of this.

System: Does reice grow in Russia?
Student: Yes, it does.
System: Why do you think rice grows in Russia?
Student: Because it has flat field and many rivers.

In this case, HSMIS can obtain a fact and a clause as follows:

```
    grow(rice, russia, true);
    grow(rice, Place, T1)::- flat-field(Place, T2), rivers(Place, T3).
```

The clauses obtained from the student are called "meta-oracles".

6.3.4 Control of the scope of the model building and topics

Domain knowledge is usually organized in a hierarchy, in which many layers of concepts(predicates) appear. When the hierarchy is deep, it is necessary to keep the scope(depth) of the hierarchy within a reasonable size to treat in a phase of tutoring. To realize this mechanism, the system has to make assumptions of validity of under the lowest predicates in the scope.

7. THEMIS

We have thus far discussed mechanisms to avoid unnecessary inconsistency in model induction. As mentioned above, however, there exist students who have contradictions in their heads. To cope with modelling of such students, the system may not avoid the inconsistency but has to model inconsistent knowledge as it is. HSMIS is capable of representing knowledge in multiple worlds with the help of ATMS. THEMIS employs MWC: Multi-World controller using ATMS as a mechanism for selecting an appropriate set of reasonable interpretations of assumptions.

7.1 MWC

Formulation of multi-world contradiction using MWC is based on the authors' speculation that humans partition their whole storage and inference spaces into multiple worlds and organize them in a discrimination tree. When solving problems they retrieve their knowledge firstly by
1) retrieving which world(concept) the given problem belongs to along a certain
discrimination structure, and then by
2) retrieving a method that contributes to the problem solving in the world
corresponding to the problem.

The first step, that is, decision on the target world, can be regarded as a search on a concept discrimination tree from its root, in which a node corresponding to a concept and each world is associated with a leaf node. The given problem is articulated into a vector of primitive attributes, which identifies the conceptual world the problem belongs to by seeking on the concept discrimination tree. To go forward through a path from one conceptual node to another requires to satisfy some conditions which characterize the destination node(world). Each single world is consistent and confusion of any two or more worlds possibly causes contradictions. The status of a student who has not yet discriminated two concepts can be modelled as not having built such discrimination conditions. Thus, the multi-world contradictions are modelled as erroneous concept discrimination tree organization shown below.

Concept discrimination trees are given in advance as a part of domain-dependent knowledge. MWC is given the whole set of worlds which are dealt with in one course of tutoring, and manages the status of each discrimination condition in the trees and sets of oracles that belong to respective worlds. MWC is able to retrieve all the clauses in all the worlds which are unifiable with a certain oracle in a world with the help of ATMS. Model diagnoses and revisions can be done on the discrimination trees. In each world, the clause level student model is inductively inferred from the oracles belonging to the world using HSMIS. It is realized by modifying the algorithm of SMDL interpreter so that a clause C in a world W is unifiable only with oracles belonging to W. Each clause level student model can be consistently inferred using such a mechanism.

The construction process of the student model that represents multi-world contradictions is as follows:

1) The system assumes a multi-world contradiction when new reliable oracles are not satisfied by a unifiable clause in the corresponding world in a reliable student model.
2) The system tests whether the oracles are satisfied by the clauses that exist in another world by visiting the world in turn in order of similarity to the correct world on the structure of the tree.
3) When a clause explaining the oracle is found in some world, discrimination conditions that contribute to differentiation of the two worlds are marked as neglected.
4) If any satisfiable worlds are not found, the system considers the situation as a single world contradiction and tries to revise the model in the correct world.

7.2 Heuristics to distinguish contradictions

One of the serious problems in modelling contradictions is to identify which type of contradiction the observed phenomenon belongs to. It is difficult for not only modelling systems but also human teachers to distinguish and detect the four types of contradictions discussed in 6.1, since all of their indications are very similar. They are triggered by a difference between the expectation of student answer deduced from the current student model and his/her actual answer. One of the goals of this research is to produce a generic and formulated modelling mechanism which is able to cope with four kinds of contradictions. Although a generic methodology to distinguish them is not fully developed, some heuristics are employed as shown below.

Assume that the reliability of each given oracle and each clause in the student model are available. Although both single world and multi-world contradictions are detectable by quite similar triggers, contradiction resolving procedures for them are quite different from each other. Single world contradictions should be resolved by revising the set of oracles or current model in general, while multi-world contradictions have to be modelled as they are. Contradiction resolution procedures for each type of single world contradiction are also a bit different, and hence detection processes of them are different from each other. In the heuristics, multi-world contradictions is first distinguished from single world contradictions. Multi-world contradictions require to revise neither oracle set nor clauses that are inconsistent with oracles, but to revise discrimination structure to permit the model to contain the inconsistency in it. Such a difference in the treatment of the two kinds of contradictions suggests the following way of discriminating them.

If either the reliability of a clause which is inconsistent with valid oracles or that of the oracles is less than a certain threshold, the inconsistency should be considered to be a single world contradiction and hence should be resolved. On the other hand, if both of the reliability is high enough, the inconsistency is considered to be a multi-world contradiction. It is not revised but put into some worlds, i.e., all the reliable data can be alive in the multi-world formulation. The following heuristics to detect contradictions of each subcategory in single world contradictions are incorporated.

The change of student's knowledge which causes type 1) of single world contradictions occurs especially right after his errors are corrected. He/she then generally changes his/her understanding from erroneous status to correct one, that is the reliability of the model is low. It is appropriate to apply revision procedures for type 1) when correct oracles are obtained right after tutoring, i.e., the system resolves the contradiction by excluding the past oracles inconsistent with correct clauses, or by asking him truth values of the oracles. The revision of oracles results in the revision of the model, i.e., erroneous clauses are dismissed and correct clauses are appended. In addition, it is available to directly ask the student if he/she has changed his/her knowledge. Independently of the correctness, generally speaking, the student is expected to have consistently applied the clauses(knowledge) that may be inconsistent with newly obtained oracles throughout a certain period, in the case that he/she makes careless mistakes which cause type 2) of single world contradictions. Thus such type of contradictions are recognized as the inconsistent oracles of low reliability and clauses of high reliability. By asking him a very similar question, the system can obtain a reliability information of the previous answer(oracle). These contradictions can be more sufficiently distinguished by introducing domain-dependent heuristics, e.g., students tend to "confuse a uniformly accelerated motion with a uniformed motion if the motion is vertical," in addition to the domain-independent heuristics explained above.

There is one more point which should be considered in designing a student modelling system. It can be assumed that there exists a student who hardly behaves consistently, because of his/her low capability or system's inappropriate selection of the level of task. It does not make sense to let such a student complete the current task. It is possible to detect such a status of the student by diagnosing the past record of acquired oracles. In such cases, the modelling system should give up modelling him/her and inform the monitor of the failure of the modelling him/her so as to let the student go back to elementary tasks.

8. BUG ANALYSES

Basically, HSMIS/THEMIS do not have a concept of bugs, so the student model module has to analyze the student model built in order to know what bugs he/she has. Bug analysis is composed of two procedures such as bug identification and bug causality analysis.

8.1 Bug identification

SMDS, a module in SMIS, is again used for bug identification. In student modelling, it is used for identifying incompleteness and incorrectness of the model using student's answers as oracles independently of they are correct or not, since its objective is to obtain a model which explains behavior of the student. In bug identification, however, the model is assumed to represent the student correctly and what we have to do is to find out bugs in it. Bugs are defined as differences between the model and the domain knowledge. So, SMDS checks the model using the answers of the domain knowledge as oracles. Thus, SMDS identifies incorrect and missing clauses and predicates(factors) in the model.

8.2 Bug causality analysis

The purpose of tutoring is to correct the bugs students have. For this purpose, the tutoring module has to know where the bugs come from, since such information helps give the student appropriate instruction. Therefore, what to do after bug identification is to identify the correct knowledge corresponding to the bug identified. This task is referred to as bug causality analysis. It is an important but difficult task in ITS. Our framework deals only with the information about the correspondence between buggy and correct clauses and discards the reason why the student comes to have them. Let us see an example shown below. The generic problem solver of this module identifies that (S1) and (S2) correspond to (E1) and (E2), respectively, and (E3) is missing in the student model.

Student model(SMDL)	Expertise knowledge(Prolog)
(S1) A::-B,C.	(E1) A:-B,D.
(S2) A::-D,E,F.	(E2) A:-D,E.
	(E3) A:-F.

The algorithm to calculate the similarity between the two rules is realized as a general mechanism according to the design philosophy described in 2.3. The basic idea behind this algorithm is that the degree of coincidence of the sets of instances derived by respective rules can be used as the similarity between the rules. By this algorithm the size of the difference set between the instance sets is reflected on the similarity and hence the semantic difference between rules is reflected on the similarity. The details of the decision procedure for the similarity are omitted here. For details, refer to [Mizoguchi, 1988]. It is designed to consider the errors which are often produced as deformation of the configuration of rules(such as exchange, insertion, and omission of predicates). For the concept with similar semantics such as "hot" and "warm", the exchange is assumed with priority.

9. EXAMPLES OF THE BEHAVIORS OF HSMIS/THEMIS

The final version of the paper will include what knowledge one has to prepare to run HSMIS/ THEMIS and how they work.

10. TUTORING MODULE

In order to show that HSMIS/THEMIS provide necessary and sufficient information to the tutoring module, this section discusses how the tutoring module in FITS uses the student model built by HSMIS/THEMIS.

10.1 Strategies

FITS has several kinds of domain-independent tutoring strategies. Although it is almost impossible to implement a tutoring module without knowing teaching material at all, it must have some property inherent to tutoring itself. An obvious example is one which gives the student correct answers immediately when he/she makes mistakes. This strategy is based on a simple mechanism for printing the file containing the correct answers, though it refers the domain-specific data in the file. A key idea of designing domain-independent tutoring strategies is to clearly distinguish between domain-dependent data or facts and the mechanisms dealing with them. FITS has 20 tutoring strategies shown below.

Explanation-based strategy
- Explanation at a deep knowledge level
- Explanation of vocabulary
- Explanation of correct knowledge
- Explanation of derivation process of examples

Hint-based strategy
- Indication of correctness of the student's solution
- Indication of incorrectness of the student's solution
- Presentation of kinds of bugs
- Presentation of the portion where bugs exist
- Presentation of an example conflicting with the student's answer
- Presentation of some examples of common factors of interest
- Presentation of some examples of different factors of interest
- Presentation of abstract examples
- Presentation of subgoals
- Presentation of an intermediate solution
- Presentation of the purpose of the examples
- Presentation of trace of the solution process
- Presentation of attributes of the example
- Presentation of the correct answer
- Suggestion of verification of the solution
- Presentation of verification process

One can synthesize many macro-strategies by combining these strategies. A macro-strategy referred to as Instance-Based strategy(IB) built in FITS is described in the following. We examined the performance of the above strategies by synthesizing the tutoring behaviors presented in Japanese typical ITSs and identified all of them can be reproduced successfully. Although evaluation of these tutoring strategies has not been done from educational point of view, the computational power and flexibility of them are shown satisfactory. Please note that one of the major objectives of the authors research, as is described in Section 3, is to explore the computational technologies for building advanced educational systems. Using these strategies, one can easily build an educational system with high reactive and adaptive behaviors.

10.2 IB and automatic problem generation

IB tries to guide the students' self-correction of their knowledge by providing them with some critical examples from which a contradiction is directly derived. If the student misses a correct clause, it gives him/her some problems from which the clause can be induced. Tutoring module contains a subsystem for generating appropriate problems in addition to a control flow of the dialog. It generates some problems according to the generate and test paradigm. Candidate problems are generated by, for example, instantiating some clauses in the expertise model. Since both the student and expertise models are executable, the engine can easily select appropriate one by comparing the answers obtained from the both.

===

INSERT Table 2

===

The operation of the strategy IB is described in the following, using the model shown in Table 2 as an example, where the student understands incorrectly the rule "if A and B then C" as "if A then C". Table 2 shows the instance (corresponds to a problem in tutoring) for A, as classified according to the truth values of the predicate B and C. pi, ri, si are the problems, for which correct responses are predicted by the model, and qi is the problems, for which incorrect responses are predicted. First, by giving qi to the student, the strategy guides him/her to recognize the incorrectness of his/her solution. Then, it helps him/her to inductively think of the missing predicate C by giving pi and having him/her consider the difference between pi and qi. For other types of bugs, those are, missing-rule, extra-rule and extra-condition, FITS has similar domain-independent strategies.

For the generation of the instance to be presented to the student, it is possible to define the problem space formally and to introduce the additional domain-dependent knowledge to it. When the correspondence between rules (the rule in the domain knowledge and the rule in the student model) for generating instances to be presented are given, SMDL interpreter returns a set of satisfying instances. The additional knowledge is introduced as the knowledge to determine whether or not each of the generated instances should be employed such as the popularity, indicating to what extent the instance is known to the ordinary students. For example, Alaska and Kiev are famous for their cold whether, and so on.

As shown in the above description of how IB works, the student model built by HSMIS provides IB with necessary and sufficient information about the student understanding state. One easily sees IB would not work without this model. In the case of THEMIS-built model, the tutoring module works in a very similar way. Suppose a student has both "if A and B then C" and "if A then C" in his/her head at the same time which HSMIS cannot model. IB works as follows:

1) Give the student a problem qi such that he/she solves it using the rule "if A then C" ("if A and B then C") which was observed before.
2) Give him/her another problem qi such that he/she solves it using "if A and Bthen C" ("if A then C").
3) Point out the contradictory answers.
4) Give him/her various levels of hints to make him/her to consider the contradiction or explanation to resolve the contradiction within him/her.

The difference between the two tutoring behaviors is that IB indicates the inconsistency between the answer of the student and that of the system, while the tutoring of THEMIS case indicates the inconsistency between two answers of the student which is a real "contradiction".

11. CONCLUSION

This paper has presented a comprehensive student modelling method based on nonmonotonic model inference in the framework of logic. Although technical details are omitted, the philosophies behind the method and conceptual structure have been discussed in detail. HSMIS/THEMIS have been fully implemented in Common ESP, an object-oriented Prolog developed by ICOT during the 5th generation computer project in Japan, on a Unix work station. The results obtained are very satisfactory in that they have shown possibilities of overcoming one of the most difficult problems in student modelling, inconsistency. The authors believe that to challenge the important but difficult problems is a good motivation of research which stimulates the research activities.

<REFERENCES>

[Chandrasekaran, 1986] Chandrasekaran, B. Generic tasks in knowledge-based reasoning: High level

building blocks for expert system design. IEEE Expert, Fall, pp. 23-30, 1986.
[de Kleer, 1987] de Kleer, J. et al.: Diagnosing multiple faults, Artificial Intelligence, Vol.32, pp.97-130, 1987.
[Ikeda, 1988a] Ikeda, M., et al.: Design of a general framework for ITS, Proc. of ITS'88, pp. 82-89, Montreal, June, 1988.
[Ikeda, 1988b] Ikeda, M. et al.: A hypothetical model inference system, Trans., on IEICE of Japan, Vol. J71-D, No. 9, pp.1761-1771, 1988(in Japanese).
[Ikeda, 1989] Ikeda, M. et al.: Student Model Description Language SMDL and Student Model Inference System SMIS, The Transactions of the Institute of Electronics, Information and Communication Engineers D-II, Vol. J72-D-II No. 1 pp. 112-120, 1989.
[Ikeda, 1993a] Ikeda, M. et al.: Nonmonotonic model inference - Proc. of IJCAI'93, Chambery, France, pp.467-473, 1993
[Ikeda, 1993b] Ikeda, M. et al.: FITS: A framework for ITS - A computational model of tutoring -, AI Technical Report, ISIR, Osaka University, AI-TR-93-5, 1993.
[Kawai, 1985] Kawai, K. et al.: A framework for intelligent CAI systems based on logic programming and inductive inference. Trans., on IPS of Japan, Vol. 26, No. 6, pp. 1089-1096, 1985(in Japanese), also appears in New Generation Computing, Vol. 5, No. 1, pp. 115-129, 1987.
[Kono, 1992] Kono, Y. et.al.: To contradict is human - Student modeling in consistency, Proc. of ITS-92, Montreal, pp.451-458, 1992.
[Kono, 1993a] Kono, Y. et al.: A modeling methods for students with contradictions, Proc. of AI-ED93, Edinburgh, 481-488, 1993.
[Kono, 1993b] Kono, Y. et al.: THEMIS: A nonmonotonic inductive student modeling system, AI Technical Report, ISIR, Osaka University, AI-TR-93-3, 1993.
[Laird, 1986] Laird, J. et al.: Universal subgoaling and chunking, Kluwer Academic Publishers, 1986.
[Mizoguchi, 1988] Mizoguchi, R., M. Ikeda and O. Kakusho. An innovative framework for intelligent tutoring systems. in P. Ercoli and R. Lewis eds. Artificial Intelligence Tools in Education. pp. 105-120, 1988.
[Mizoguchi, 1990] Mizoguchi, R. et al. A generic framework for ITS and its evaluation, Proc. of International Conference on ARCE, Tokyo, pp.303-312, 1990.
[Mizoguchi, 1993] Mizoguchi, R.: Student modelling in ITS, Tutorial note of ICCE93, Taipei, 1993.
[Shapiro, 1982] Shapiro, Y.: Algorithmic program debugging, MIT Press, 1982.

Table 1: Definition of $\wedge$ and $\vee$.

(a) $\wedge$ operator

$\wedge$	true	unk.	false	fail
true	true	unk.	false	fail
unk.	unk.	unk.	false	fail
false	false	false	false	fail
fail	fail	fail	fail	fail

(b) $\vee$ operator

$\vee$	true	unk.	false	fail
true	true	true	true	true
unk.	true	unk.	unk.	fail
false	true	unk.	false	fail
fail	true	fail	fail	fail

Table 2: Problems generated by the tutoring module

B	C	Expertise A (A:-B,C)	Student A (A::-B)	instances
true	true	true	true	$p_1, p_2, \cdots$
true	**false**	**false**	**true**	$q_1, q_2, \cdots$
false	true	false	false	$r_1, r_2, \cdots$
false	false	false	false	$s_1, s_2, \cdots,$