Nonmonotonic Student Model Inference System

Riichiro MIZOGUCHI, Mitsuru IKEDA, and Yasuyuki KONO

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

Abstract: A nonmonotonic student modeling system THEMIS based on logic programming paradigm is described. A student model description language named SMDL which takes four truth values is designed to capture the student behavior. By asking questions when necessary, THEMIS builds a student model as an SMDL program which explains the behaviors of the student. It is designed on the basis of close analysis of inconsistencies which are classified into four categories according to the causes of them. Further, we identified they are grouped into two groups: one should be resolved and the other should be modeled as it is. THEMIS can cope with all kinds of inconsistencies appearing during the student modeling.THEMIS can model students with inconsistent knowledge in their heads as it is in order to give a tutoring module information necessary for performing Socratic tutoring. THEMIS is implemented in CESP:Common ESP, a logic-based object-oriented language on a Unix workstation.

1. Introduction

The authors have been involved in ITS research for several years(Kawai, Mizoguchi, Kakusho, & Toyoda, 1985)(Mizoguchi, Ikeda, & Kakusho, 1988)(Ikeda, Mizoguchi, & Kakusho, 1988a)(Ikeda, Mizoguchi, & Kakusho, 1988b)(Ikeda, Mizoguchi, & Kakusho, 1989)(Mizoguchi & Ikeda, 1990)(Kono, Ikeda, & Mizoguchi, 1992)(Ikeda, Kono, & Mizoguchi, 1993)(Ikeda & Mizoguchi, 1993)(Kono, Ikeda, & Mizoguchi, 1993a) (Kono, Ikeda, & Mizoguchi, 1993b). This paper reviews their research activities on student modeling 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 modeling module plays a crucial role. Section 3 discusses the design philosophies of the student modeling system THEMIS. The rests of the paper present conceptual level description of THEMIS. Readers interested in the technical details are advised to refer to (Ikeda, Kono & Mizoguchi, 1993)(Kono, Ikeda & Mizoguchi, 1993b).[1]

2. Overview of the Research

2.1 Research objectives

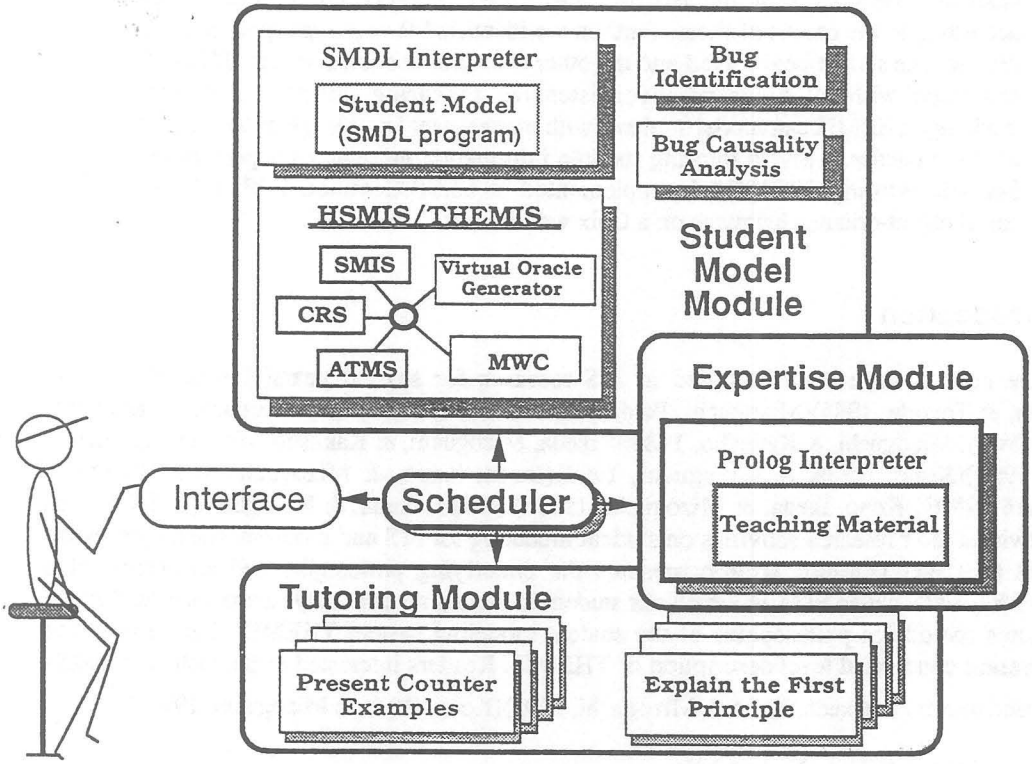
Our research has been conducted aiming at the following two major goals:
1) to reveal inherent computational structure of ITS, and

[1] Comparison between the proposed method and related work is done in Chapter 3.

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 performance 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 many 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 modeling.



SMDL : *Student Model Description Language*
HSMIS: *Hypothetical SMIS*
SMIS : *Student Model Inference System*
ATMS : *Assumption-based Truth Maintenance System*
CRS : *Contradiction Resolving System*
MWC: *Multi-World Controller*

Fig. 1 Block diagram of FITS.

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. As is shown in Fig. 1, FITS is composed of seven primitive modules such as student model interpreter(SMDL interpreter), HSMIS/THEMIS, Bug identification, Bug causality analysis, Expertise, Tutoring strategy, 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 modeling language called SMDL(Student Model Description Language) based on Prolog (Mizoguchi, Ikeda, & Kakusho, 1988). In our framework, the student model is represented as an SMDL program, so modeling 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 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 necessarily use domain-specific data and facts in explanation and other interactions. By domain-independent tutoring strategy, we do not mean no domain-specific data is used but mean their mechanisms are domain-independent. In fact, a number of domain-independent strategies are implemented in FITS. 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 the tutoring strategies and the scheduling mechanism, refer to(Mizoguchi & Ikeda, 1990)(Ikeda & Mizoguchi, 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 attracts much attention which can be building blocks of expert systems(Chandrasekaran, 1986). By generic tasks, we mean domain-independent but task-dependent chunk of control structures which reflect inherent problem solving structures of corresponding tasks. In order to obtain a generic framework, all the

modules have to be designed independently of a teaching material. On the basis of the generic task theory, 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 which contributes to clarification of the characteristics of functions of the module.

3. Design Philosophy of THEMIS

Discussions on student modeling have been made from two major view points: Computational and cognitive ones. There is no need to stress the importance of cognitive aspects of student modeling. 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.

Modeling human understanding is a really challenging and ambitious problem. Although we know much about 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. We believe technological progress provides the AI in education community with powerful tools for designing and building educational systems.

There are several dimensions for characterizing student modeling 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

Modeling 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 modeling methods usually build more comprehensive models than analytic methods.

Furthermore, inductive modeling 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, modeling 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

Student modeling mechanisms should be based on a sound theoretical foundation. It contributes to both clarification of the inherent property of student modeling problems and to articulation of the scalability and reusability of the proposed mechanisms. Efficient domain-specific modeling methods are of course important. However, we do need general methodologies and theories of student modeling. In order to develop such a method supported by a firm theoretical background, model inference in the logic programming paradigm is employed in THEMIS.

4) Inconsistency

Viewing student modeling 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 modeling is essentially non-monotonic. The modeling 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 fine-grained primitives of the domain knowledge. While the former requires a lot of effort to enumerate and analyze bugs but modeling is generally efficient, the latter is free from bug analysis and has high generality but modeling 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, modeling from one observation and quick didactic feedback can realize neither comprehensive nor well-organized tutoring. THEMIS employs modeling from multiple observations to realize sophisticated and comprehensive tutoring.

7) Link to how to use the model

Tutoring behavior depends largely on the 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 modeling problem is "intractable". Therefore, designers of student modeling 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. A student model should not

be designed independently of tutoring module's requirements. The tutoring module in FITS is based on sophisticated hint-giving strategies 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 modeling methods proposed to date do not ask the students questions during the course of modeling, 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 modeling 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 modeling 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 our purpose because it only takes two kinds of truth values. Let us consider how many truth values are necessary. First of all, the modeling 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 modeling language has to take two 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 modeling language which is executable and can simulate the problem solving behavior of the student(Mizoguchi, Ikeda, & Kakusho, 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". 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.

Table 1 Definition of AND and OR operations.

(a) AND operator

∧	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) OR operator

∨	true	unk.	false	fail
true	true	true	true	true
unk.	true	unk.	unk.	fail
false	true	unk.	false	fail
fail	true	fail	fail	fail

5. SMIS: Student Model Inference System[2]

SMIS is an inductive inference system for SMDL. It is an extended version of MIS(Shapiro, 1982)(A brief explanation of MIS is found in section 5.3 in Chapter 3). 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[3] 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

[2] This chapter assumes the readers understand the content of sections 5.2 and 5.3 in Chapter 3.

[3] The meaning of the incorrectness of the model is discussed in section 5.3 in Chapter 3.

traces the computation process of the current model and checks the results against student's answers. For example, the following clause is deleted when oracles such as grow(rice, japan, false), moderate(japan, true), and wet(japan, true) are obtained:

grow(Plant, Places, T)::- moderate(Place, T1), wet(Place, T2),

since the fact derived by the clause, grow(rice, japan, true) conflicts with the oracle grow(rice, japan, false).

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 the oracle and some others, respectively. 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 modeling method with SMIS is sufficiently high, since it can build any models which can be described in SMDL.

6. HSMIS: Hypothetical SMIS

The inductive inference algorithm of SMIS is based on the assumption that all the oracles are consistent, that is, student's answers 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 modeling 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 modeling process is classified into the following four categories according to their causes:

1) Oracle contradiction caused by the change of students' understanding:

Learning process of a student is essentially acquisition of new knowledge, which necessarily causes the 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 one. Therefore, he/she answers "yes" to a question to which he/she had answered "no".

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 generates contradictory oracles.

4) Assumption contradiction in modeling:

Inductive inference is essentially a hypothetical process even if data are consistent, 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 modeling 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 because the student should be modelled in a single consistent world to avoid them. And we call the latter multi-world contradictions because 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, 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. A set of assumptions that the individual derived data depend on can be calculated by tracing up the chain of justifications starting from the derived data. 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.

6.3 Overview

HSMIS consists of SMIS, ATMS, Virtual oracle generator(explained in 6.4.2), 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 necessarily appears among oracles. Virtual oracle generator is responsible for decreasing the questions made by the SMIS by generating virtual student answers based on the reliability of the student without asking the student questions. CRS resolves the inconsistency by changing the environment in which the model inference is done. Inconsistencies(contradictions) in HSMIS is defined as rules according to the causes described above(See (Kono, Ikeda, & Mizoguchi, 1993b) for details).

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 stores it in the nogood record.
3) SMIS asks CRS to resolve the inconsistency.

4) According to the causes of the 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.

6.4 Controlling the modeling 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 HSMIS.

6.4.1 Heuristics for student modeling

A refinement graph spanned by refinement operators defines the search space of each clause of interest. However, it 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 the 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 a bit more general clauses than these bugs, which often makes the search very efficient. When it fails, it resumes the search from the original root. Therefore, introduction of heuristics does not loose any generality. Note 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 modeling efficiency.

6.4.2 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 completely"***. 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. Virtual oracles are the answers by the domain knowledge. They are always "correct" but does not always coincide with those of the students. 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 a contradiction occurs, the clauses supported by virtual oracles are the first candidate to withdraw as well as the virtual oracles.

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(virtual oracles). When the assumed 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 manages them with the aid of ATMS. This mechanism helps decrease the questions given by the HSMIS.

6.4.3 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 rice 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".

6.4.4 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. HSMIS is equipped with a mechanism for supporting this.

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 modeling of such students, the system may not avoid the inconsistency but has to model inconsistent knowledge as it is. THEMIS is capable of representing knowledge in multiple worlds with the help of ATMS. It 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 whose root corresponds to the general class of the problems of interest, intermediate nodes to concepts which characterize the descendant nodes, and leaves to respective worlds in which problem solving takes place. 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.

The first step, that is, decision on the target world, can be regarded as a search on a concept discrimination tree from its root. The given problem is represented in a vector of primitive attributes, which identifies the conceptual world the problem belongs to. 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 in his/her head. Thus, the multi-world contradictions are modelled as erroneous concept discrimination tree organization. Clause level modeling which is one discussed in the previous sections is done in each simple world using HSMIS.

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. Diagnosis and revision of the model can be done on the discrimination tree. 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.

Let us summarize the construction process of the student model that represents multi-world contradictions. When the current model explains the observed oracle, the system does nothing. Otherwise, it tries to explain it according to the following procedure:

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. Otherwise, it considers single world contradiction.
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.

THEMIS calculates the reliability of each clause in its clause(method) level student model. The calculation is done by referring to various kinds of information, i.e., number of top-level traces that justify the clause, whether the oracles which consist of each top-level trace of the clause are correct answers or not, how old the oracles are, etc. Reliability of an oracle is calculated by referring to the time the student used before answering the question, by comparing it with answers to questions similar to the current question, etc.

7.2 Heuristics to distinguish contradictions

One of the serious problems in modeling contradictions is to identify which type of contradiction the observed phenomenon belongs to. In theory, a contradiction except type 4 discussed in 6.1 is found after failing to build a model covering all the oracles observed thus far. But it usually takes much time until detecting the failure because it is detected after all the resources are exhausted. Therefore, heuristics are introduced for the detection discussed below. Note that it is difficult not only for modeling systems but also for human teachers to detect and distinguish the four types of contradictions, 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.

Although both single world and multi-world contradictions are detected by 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 the current model, 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 as shown in 7.1. 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 discrimination.

Assume that the reliability scores of each given oracle and each clause in the student model are available. Suppose a mismatch between the oracle and the expected truth value is found. 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 it 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 are heuristics to detect contradictions of each subcategory in single world contradictions.

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 a tutoring action, 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 again. 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.

In the case that he/she makes careless mistakes which cause type 2) of single world contradictions, reliability of such an oracle should be low and that of the clause conflicting with the oracle is high, since the student must have used the knowledge for solving several problems before the cereless mistake. This is based on an assumption that the students make careless mistakes not so often. 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.

Needless to say, when the reliability of the model is low and that of the oracle is high, which is the most common situation, it is considered as a modeling contradiction in which the model is revised according to the normal SMIS modeling procedures.

There is one more issue which should be considered in designing a student modeling 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 modeling system should give up modeling him/her and inform the monitor of the failure of the modeling 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 modeling 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 modeling, 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 the module has to do is to find out bugs in it. Bugs are defined as the deviations from 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. In this module, SMDL program is the search space and SMDS is a generic problem solver.

8.2 Bug causality analysis

One of the purposes of tutoring is to correct the bugs students have. To do this, the tutoring module has to know where the bugs come from, since such information helps give the student appropriate instructions. Therefore, what to do after bug identification is to identify the correct knowledge corresponding to the buggy knowledge identified. This task is referred to as bug causality analysis. Strictly speaking, 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)[4]
(S1) A::-B,C.	(E1) A:-B,D.
(S2) A::-D,E,F.	(E2) A:-D,E.
	(E3) A:-F.

The algorithm for identification of the clause correspondence is based on the similarity between the clauses which 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 clauses can be used as the similarity between the clauses. By this algorithm the size of the difference set between the instance sets is reflected on the similarity and hence the semantic difference between clauses is reflected on the similarity. The details of the decision procedure for the similarity are omitted here. For details, refer to (Mizoguchi, Ikeda, & Kakusho, 1988). It is designed to consider the errors which are often produced as deformation of the configuration of clauses(such as exchange, insertion, and omission of predicates).

9. Examples of the Behaviors of HSMIS

Leu us show an example of how HSMIS work below. Sentences in bigger fonts are outputs from the system, clauses in boxes are the ones in the current student model, sentences in italic font are for explanations and others are internal expressions of justifications given from SMIS to ATMS and of oracles corresponding to the student's answers in which 'in' and 'out' denote the data is currently believed and not believed, respectively.

[4] Arguments are omitted for simplicity.

Does rice grow in Japan? → C1: grow(Place,T) ::- temp(Place,T'), soil(Place,T").

yes. o1: grow(japan,T_1), yes.
v_o2: temp(japan,T_2), yes.
v_o3: soil(japan,T_3), yes.

Because o1 is a correct answer, the correct clause C1 is added to the model. Then, v_o2 and v_o3, which are correct answers, are generated by Oracle-generator. They are justified by trust(C1) and oracle(o1). ATMS is informed of the following assumptions and justifications.

Assume: oracle(o1) , trust(C1), Ω-consistent(C1),general(C1)
Justifications:
trust(C1), oracle(o1) ⇒ v_oracle(v_o2)
trust(C1), oracle(o1) ⇒ v_oracle(v_o3)
oracle(o1), v_oracle(v_o2), v_oracle(v_o3) ⇒cover(C1,o1).
oracle(o1), Ω-consistent(C1), cover(C1,o1), general(C1)⇒model(C1)

Does rice grow in Kiev? → C2: grow(Place,T)::- soil(Place,T').

yes. o4: grow(kiev,T_4), yes.
v_o5: soil(kiev,T_5), yes.

A buggy clause C2 is added to the model, because o4 is a typical incorrect answer(Kiev is too cold for rice grow). A virtual oracle v_o5, which is justified by trust(C2) and oracle(o4), is generated by Oracle-generator. Since the clause C1 is a descendent node of C2 in the refinement graph, model(C1) becomes 'out' by outing general(C1). Then the status of v_o2 and v_o3 becomes 'out'. However, v_o3 becomes 'in', because it is justified by trust(C2) and oracle(o1)

Assumeptions oracle(o4) , trust(C2), Ω-consistent(C2),general(C2)
Justifications:
trust(C2), oracle(o1) ⇒ v_oracle(v_o3)
trust(C2), oracle(o4) ⇒ v_oracle(v_o5)
oracle(o4), v_oracle(v_o5) ⇒cover(C2,o4).
oracle(o4), Ω-consistent(C2), cover(C2,o4), general(C2)⇒model(C2)
oracle environment
in o1:<grow(japan,T_1),yes. >
out v_o2:<temp(japan,T_2), yes. >
in v_o3:<soil(japan,T_3), yes. >
in o4:<grow(kiev,T_4), yes. >
in v_o5:<soil(kiev,T_5), yes. >

Does rice grow in Moscow?
no. o6:grow(moscow,T_6), no.
Is Soil of Moscow suitable for rice to grow?
yes. o7:soil(moscow,T_7), yes.

SMDS: C2 is refuted by o6 and o7; Refutation: grow(moscow, false)::-soil(moscow,true)

C2 is removed because SMDS found out its refutation: (grow(moscow,no)::- soil(moscow,yes)). A contradiction is found between Ω-refutation(C2) and Ω-consistent(C2). CRS resolves this by outing the default assumption Ω-consistent(C2). Then model(C2), virtual oracles v_o3 and v_o5 becomes 'out'. HSMIS starts to search for a clause which covers o4,

Justifications:
oracle(o6), oracle(o7)⇒ Ω-refutation(C2).
oracle environment
in o1:<grow(japan,T1),yes. >
out v_o2:<temp(japan,T2), yes. >
out v_o3:<soil(japan,T3), yes. >
in o4:<grow(kiev,T4), yes. >
out v_o5:<soil(kiev,T5), yes. >
in o6:<grow(moscow,T6), no. >
in o7:<soil(moscow,T7), yes. >

Is soil of Kiev suitable for rice to grow?

yes. o5:soil(kiev,T_5), yes. (confirmed by the student)

Is Temperature of Kiev suitable for rice to grow?

no. o8: temp(kiev,T_8), no.

SMIS: C3:grow(Place,T):-temp(Place,T) cannot cover grow(kiev,yes).

Contradiction is found, because HSMIS has failed to search for a cover of grow(kiev,yes). CRS changes the oracle environment to resolve the contradiction based on the give-a-priority-to-correct-answer heuristics In this case, o4 becomes 'out' and o9, whose truth value is opposite to o4, is believed according to give-priority-to-correct-answer heuristic. Finally, C1 is added to the current model (model(C1) becomes 'in' again) in the new consistent oracle environment.

oracle environment

in o1:<grow(japan,T_1),yes. >
in v_o2:<temp(japan,T_2), yes. >
in v_o3:<soil(japan,T_3), yes. >
out o4:<grow(kiev,T_4), yes. >
in o5:<soil(kiev,T_5), yes. >
in o6:<grow(moscow,T_6), no. >
in o7:<soil(moscow,T_7), yes. >
in o8:<temp(kiev,T_8), no. >
in o9:<grow(kiev,T_9), no. >

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 in tutoring itself. An obvious example is one which gives the student correct answers immediately when he/she answers an incorrect answer. This strategy is based on a simple generic 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 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(See Fig. 2). We examined the performance of the above strategies by synthesizing the tutoring behaviors of typical ITSs developed in Japan 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. 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 tutoring 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.

The operation of the strategy IB is described in the following, using the model shown in Table 2 as an example, where the student incorrectly understands the rule(clause) "if B and C then A" as "if B then A". Table 2 shows the instances (correspond to problems in tutoring) for A, as classified according to the truth values of the predicate B and C. p_i , r_i, s_i are the problems, for which correct responses are predicted by the model, and q_i is the problems, for which incorrect

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	$s_1, s_2, \cdots,$

(a) Does rice grow in Tokyo ?
Answer ? ***yes.***
Does rice grow in Kiev ?
Answer ? ***yes.***
Does rice grow in Bei-jing ?
Answer ? ***yes.***
(b) You made some mistakes. For example, Rice does not grow in Kiev. Do you understand the reason?
Answer ? ***no.***
(c) You made 2 mistakes. Now let us consider the problem.
c-1 Rice grows in the following places :
Tokyo Nan-ching Jakarta
c-2 Rice dose not grow in the following places :
Kiev Bei-jing Detroit
Do you understand differences between the above two sets of examples?
Answer ? ***no.***
(d) Can you think of a common factor which does not hold among Kiev, Bei-jing and Detroit ?
Answer ? ***no.***
(e) All the places in c-2 satisfy soil condition, lay condition and irrigation condition. But rice does not grow there, since they do not satisfy another factor you miss out.
(f) Please select a description from (1)-(3) which explains the difference
(1) The temperature of Tokyo is suitable for rice to grow but that of Kiev is not.
(2) People in Tokyo live on rice but people in Kiev does not.
(3) Tokyo is not inland area but Kiev is.
Answer ? ***2.***
It's not good.
.....

Fig. 2 Sample dialog controlled by macro-strategy IB.

responses are predicted. First, by giving q_i 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 p_i and having him/her consider the difference between p_i and q_i . 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 a pair of rules (one in the domain knowledge and one in the student model) are given, SMDL interpreter returns a set of satisfying instances. Additional domain knowledge is introduced to select appropriate instances. An example is typicality or popularity of them indicating to what extent the instances are 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 B and C then A" and "if B then A" in his/her head at the same time and uses them exclusively, which HSMIS cannot model. IB works as follows:

1) Give the student a problem q_i such that he/she solves it using the rule "if B then A" ("if B and C then A") which was observed before.
2) Give him/her another problem q_j such that he/she solves it using "if B and C then A" ("if B then A").
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 above 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". Needless to say, the THEMIS case can give greater impact to the students than IB.

11. Domain Knowledge Required

We have thus far discussed domain-independent inference mechanisms for student model building, bug analysis and tutoring strategies. Although the mechanisms are domain-independent, they require some domain-dependent knowledge. This sub-section summarizes what kinds of domain-dependent knowledge are necessary for making the mechanisms operational.

11.1 THEMIS

THEMIS requires the following five kinds of knowledge:

(1) Domain knowledge

This knowledge apparently necessary knowledge for tutoring, since it is the knowledge for the students to understand. From the student model building point of view, however, the domain knowledge helps the modeling process by providing THEMIS with correct knowledge. It is used by oracle generator to generate default correct answer of the student(see 6.4.2), bug identification module to identify bugs of the student model built(see 8.1), bug causality analysis module to measure the similarity between the clauses in the student and domain models(see 8.2). Needless to say, the domain knowledge is used for evaluating the student answers and for generating problems and hits(see 10.2). It is represented in Prolog clauses and facts.

(2) Primitive concepts

The refinement operators(see Section 5) require some sets of primitive concepts by which the search space for each clause to be added to the current model is specified. In order to cover a goal grow(rice, japan, true), for example, THEMIS requires the predicate grow(X,Y,T) and candidate knowledge(predicates) used for its body such as suitable_temp(X,Y,T), suitable_soil(X,Y,T), suitable_lay(X,Y,T), and has_irrigation(X,T) as the correct knowledge and wet(X,T), warm(X,T), etc. as incorrect ones. For each clause, a set of predicates as shown above are required.

(3) Plausible bugs

As was described earlier, THEMIS does not know bugs of the domain under consideration. But, if it is given plausible or typical buggy knowledge specific to the domain, it can build the student model very efficiently when modeling such students who have the bugs similar to those(see 6.4.1). For example, concerning the grow predicate, a typical buggy clause which could be given is

```
Buggy_clause( (grow(X,Y,T)::- suitable_temp(X,Y,T1), suitable_soil(X,Y,T2),
                has_irrigation(X,T3)).
```

Note here again that this is not necessary information for THEMIS which can perform modeling task without such knowledge.

(4) Multi-world hierarchy

MWC(see 7.1) requires a hierarchy of important concepts which characterize all the worlds in which problem solving takes place.

(5) Natural language expressions

In order to translate internal representation to natural language, interface module requires information about the sentence patterns corresponding to all the clauses and facts. Typical example is "X grows in Y" for the predicate, grow(X,Y,true).

11.2 Tutoring strategy module

This module apparently requires many kinds of facts used as hints as well as text patterns for explanation. One to note is, as is mentioned in 10.2, the typicality or popularity of the instances. For example, Cairo and Bangkok are popular in their hot and humid weather, etc. These kinds of knowledge are the domain-specific knowledge necessary for this module.

When all the above domain-dependent knowledge is given to THEMIS and tutoring module, they perform their tasks successfully.

12. Conclusion

This paper has presented a comprehensive student modeling method based on nonmonotonic model inference in the framework of predicate logic. Although technical details are omitted, the philosophies behind the method and conceptual structure have been discussed in detail. HSMIS/THEMIS is 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 a possibility of overcoming one of the most difficult problems in student modeling, inconsistency. The authors believe that to challenge important but difficult problems is a good motivation of research which stimulates the research activities.

<REFERENCES>

Chandrasekaran, B. (1986). Generic tasks in knowledge-based reasoning: High level building blocks for expert system design. IEEE Expert, Fall, 23-30.

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

Ikeda, M., Mizoguchi, R. & Kakusho O. (1988a). Design of a general framework for ITS, Proc. of ITS'88, Montreal, 82-89.

Ikeda, M., Mizoguchi, R. & Kakusho O. (1988b). A hypothetical model inference system, Trans., on IEICE of Japan, Vol. J71-D, No. 9, 1761-1771(in Japanese).

Ikeda, M., Mizoguchi, R. & Kakusho O. (1989). 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, 112-120(in Japanese).

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

Ikeda, M. & Mizoguchi, R. (1993). FITS: A framework for ITS - A computational model of tutoring -, AI Technical Report, ISIR, Osaka University, AI-TR-93-5.

Kawai, K., Mizoguchi, R., Kakusho, O, & Toyoda, J. (1985). A framework for intelligent CAI systems based on logic programming and inductive inference. Trans., on IPS of Japan, Vol.

26, No. 6, 1089-1096(in Japanese), also appears in New Generation Computing, Vol. 5, No. 1, 115-129, 1987.

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.

Laird, J., Rosenbloom, P. & Newell, A. Universal subgoaling and chunking, Kluwer Academic Publishers, 1986.

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.

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