Ph.D. proposal, Department of Computer Sciences, University of Texas at Austin, June 1993.
Also appears as Technical Report AI 94-215, Artificial Intelligence Laboratory,
University of Texas at Austin, February, 1994.

Learning to Model Students: Using Theory Refinement to Detect Misconceptions †

Paul T. Baffes
Department of Computer Sciences
University of Texas
Austin, TX 78712
(512) 471-9589
baffes@cs.utexas.edu

June 15, 1993

Presented: June 17, 1993

Abstract

A new student modeling system called ASSERT is described. ASSERT is a general purpose algorithm which uses domain independent techniques to perform student modeling and to automatically construct libraries of common bugs. For its modeling component, ASSERT uses a machine learning technique which is an extension of the EITHER theory refinement algorithm. A common library of bugs is constructed by extracting commonalities across multiple student models. Initial experimental data suggests that ASSERT is a more effective modeling system than other techniques previously explored, and that the automatic bug library construction significantly enhances subsequent modeling efforts.

1 Introduction

Almost since the advent of the computer age, researchers have recognized the computer's enormous potential as an educational aid. Early efforts to use computers as educational tools resulted in a paradigm now generally known as *computer aided instruction* (CAI). Such programs are used to automate the presentation of well prepared material to a student. In a typical CAI program, this presentation takes on a form similar to a book, with the additional ability to present different sections of material or different levels of detail based upon choices made by the user. The construction of an effective CAI program requires the author to anticipate student reactions so that appropriate choices can be encoded in the program. Knowledge of the educational task is thus *external* to a CAI program; the author

†This research was supported by the NASA Graduate Student Researchers Program under grant number NGT-50732, the National Science Foundation under grant IRI-9102926, and a grant from the Texas Advanced Research Program under grant 003658144.

uses his or her expertise to predetermine the choices and explanations which will be seen by the student.

In the 1970s criticisms of the CAI approach began to emerge. Several researchers, most notably Carbonell (Carbonell, 1970) pointed out that CAI could not be truly responsive to individual student needs until a knowledge of the domain equivalent to that possessed by the educator was encoded in the system. In short, a reactive system needed to be able to draw conclusions based upon interactions with the student similar to those made by a teacher. Since Carbonell's work, multiple efforts to construct *intelligent* CAI (ICAI), also called *intelligent tutoring systems* (ITS), have produced a variety of new techniques. In all of these, the method used to represent the knowledge of the student has played a major role. Such representations are generally referred to as *student models*.

Student modeling research has taken a variety of forms from overlay models (Carr and Goldstein, 1977) to rule-based representations (Young and O'Shea, 1981; Sleeman and Smith, 1981) to frame-based methods (Rich, 1990). The disadvantage of most of these techniques is their inability to deal with novel student errors. Additionally, they can require a great deal of effort to construct. There has also been some debate over the utility of student models, with early studies indicating ITS and CAI systems to be educationally equivalent, though the most recent results show ITS methods to be superior (Nicolson, 1992). If student modeling techniques are to be useful, then, they must be flexible enough to handle novel errors, simpler to construct, and easily tested so that we may gauge the utility of the models produced. These three essential features are precisely what another area of artificial intelligence research, machine learning, can bring to student modeling.

Our view of student modeling as a learning process is based upon the observation that a teacher's ability to model students improves with time. Presumably, this ability is enhanced as the individual becomes more familiar with how students react to the material being presented. At the very least, one should be able to take advantage of any trends which become apparent across multiple students and note them as common misconceptions. Once detected, such trends allow future students to be modeled more effectively since the teacher can anticipate problem areas which are likely to arise. Yet to date no modeling system has attempted to automatically detect such trends for use in subsequent modeling efforts. The claim made here is that a teacher's skill improves over time in part because he or she *learns to become a more effective modeler*, and that this extra learning capability should be an integral part of the student modeling process.

The research presented here outlines a new technique for using machine learning methods to construct student models for intelligent tutoring systems. Two key features distinguish this work from previous student modeling efforts. First, the method presented incorporates a learning strategy known as *theory refinement* which is shown to be a more powerful technique than previous machine learning algorithms used for student modeling. Second, an additional learning cycle is described which can be shown to increase the effectiveness of subsequent modeling efforts. The result is a meta-learning or *learning to model* paradigm that can automatically detect trends in student behavior which are then used to update the modeling process.

After a background on student modeling and machine learning is presented in Sections 2.1 and 2.2, Section 2.3 shows how the general notion of theory refinement can be used to construct student models. Section 2.4 then outlines how trends across multiple students can be detected and fed back into the student modeling process. Section 3 describes the specific theory refinement algorithm used in more detail and in Section 4 our learning to model paradigm called ASSERT (Acquiring Stereotypical Student Errors using Refinement of Theories) is presented. Section 5 describes preliminary experimental evaluations of ASSERT using both artificially generated and real student data. Finally, the last two sections compare ASSERT to other adaptive modeling efforts and propose specific extensions to enhance ASSERT's capabilities.

2 Background

2.1 Student Modeling in ICAI Research

One of the most important components of an ICAI system is the student model (Wenger, 1987). Some researchers have argued (Carbonell, 1970; Laubsch, 1975) that the effectiveness of an ICAI system depends heavily upon its student modeling component. Without the flexibility to model novel student errors, ICAI systems will not progress much beyond electronic page turners with canned responses tuned to the average student.

Over the last two decades, several techniques for student modeling have been developed. One method, called *overlay* modeling (Carr and Goldstein, 1977), assumes a student's knowledge is always a subset of the correct domain knowledge. As the student performs actions which illustrate that he or she understands particular elements of the domain knowledge, these are marked in the overlay model. Unmarked elements of the model can be used to focus tutoring on problem areas for the student, or to ensure full coverage of the domain. While simple to implement, this method is incapable of capturing misconceptions, or *bugs*, that represent faulty student knowledge.

To capture such misconceptions, other researchers (Brown and Burton, 1978; Burton, 1982; Brown and VanLehn, 1980; Sleeman and Smith, 1981) have focused on constructing databases of student misconceptions typically termed *bug libraries*. In these approaches, models are built by matching student behavior against a catalog of bugs which are preconstructed by hand through an analysis of student errors. Typically, such catalogs are either difficult to construct or fail to cover a wide enough range of behaviors. As with overlay models, the static nature of bug libraries renders them incapable of modeling unanticipated student problems.

A third method of student modeling attempts to overcome this limitation either by patching the bug library dynamically (Sleeman et al., 1990) or by modeling student misconceptions from scratch (Langley et al., 1984; Ohlsson and Langley, 1985). Here a machine learning technique called *induction* is used to fashion unique elements of the student model from examples of the student's behavior. While this provides the flexibility to capture novel bugs, in general, accurate induction requires a large number of such examples. Moreover, these approaches do not take advantage of any trends which occur across multiple students. To solve these problems, one needs a different learning mechanism which works with fewer examples than induction, can make use of any preexisting knowledge about student behavior, and can use the results of previous modeling efforts to update the modeling process.

2.2 From Induction to Theory Refinement

Over the past decade, a number of machine learning algorithms have been developed which can induce a classification system from a set of training examples (Quinlan, 1986; Michalski, 1983; Mitchell, 1982; Rumelhart et al., 1986). The examples are typically presented in the form of input-output pairs, where the input is a collection of values for features of the domain, and the output represents the category corresponding to the input. In the typical classification problem, the task of the induction algorithm is to learn, from the examples, some function which will produce the correct output category for any given set of inputs. The learned system can be thought of as a function which maps inputs to outputs corresponding to the pattern implicit in the training examples.

Casting the student modeling task as a machine learning problem is straightforward. Examples can be constructed by linking inputs taken as test problem specifications with the corresponding output provided by the student. When given to an inductive learner, the resulting system produced is a model of the student's problem solving ability. Given new test problems, the system will tend to produce the same answers as the student. Unfortunately, induction techniques typically require a large number of examples to produce accurate models and fail to provide an obvious means of improving the modeling process using the results of previous modeling efforts.

A new generation of more effective machine learning algorithms have been developed which combine induction with other machine learning techniques. These methods use two inputs, examples plus an initial domain theory (Ginsberg, 1990; Ourston and Mooney, 1990; Craw and Sleeman, 1990; Towell and Shavlik, 1991). Such learners are termed *theory refinement* systems since they take an input specification (called the *theory*) and produce a revised version of that specification which is consistent with the examples. The idea is one of refinement; the learner starts with an initial theory that is incorrect or incomplete, and modifies it to fit a set of data. For example, the EITHER system (Ourston and Mooney, 1990) alters an initially incomplete or incorrect rule base by modifying or deleting existing rules or by adding new rules until the rule base is consistent with the input examples. As with other theory refinement systems, EITHER has been shown to be much more effective at learning concepts than induction alone when given an approximately correct theory. In general this is because theory refinement need only infer the differences between its input theory and the correct theory, whereas induction must learn everything from scratch. The closer the original theory is to the desired target, the easier it is for systems like EITHER to produce accurate results.

Theory refinement, then, shows promise as a modeling technique. It is more effective than induction and can make use of potentially available information in the form of an initial theory. This ability to work with an initial theory is central to the new ideas presented here. As shall be seen in the following sections, it is the mechanism by which the modeler focuses its learning to the knowledge being presented and the method through which previous learning results are fed back into future modeling efforts.

2.3 Using Theory Refinement for Student Modeling

Thus far, theory refinement has been presented as a technique for fixing errors present in an incomplete or incorrect theory by evaluating example data. In principle, however, there is no reason why one cannot use theory refinement backwards; i.e., to purposely *introduce* errors into a theory. This is precisely how theory refinement can be used to model students. Starting with *correct* knowledge of a set of concepts, one models the student by introducing *errors* until the system matches the student's performance. If one sets the initial theory to a model of ideal student behavior (i.e., the knowledge the tutoring system wants to impart to the student), then the transformations made in creating the student model show where the student has misconceptions that caused him or her to deviate from the correct knowledge. The initial theory provides a focus for the modeling task, which becomes a search for refinements that will transform the input theory to a theory which mimics the student's behavior.

Perhaps more importantly, theory refinement techniques also offer a means of using any preexisting knowledge directly as part of the modeling process. One straightforward way to do this is to incorporate such knowledge as part of the initial theory. For example, if a set of common misconceptions is known to exist, it can be used directly by making the corresponding changes to the correct domain theory. The result is a theory which is likely to be closer to the average student, giving theory refinement an advantage by starting it closer to its target. As before, a change in any component of the initial theory that was also part of the correct theory is considered a misconception. In addition, any common bug that remains *un*changed is also considered a misconception. Thus theory refinement can be used to model students with or without the benefits of any known student errors.

Although theory refinement has been described as a process which modifies a set of rules, it should be pointed out that theory refinement in general is not limited to a rule-based knowledge representation. It should also be noted that neither theory refinement nor student modeling is limited to a particular type of problem domain. Most student modeling tasks have focused on procedural problem solving domains such as geometry (Anderson et al., 1985), subtraction (Brown and Burton, 1978; Burton, 1982; Sleeman and Smith, 1981; Langley et al., 1984), or writing computer programs (Johnson, 1986; Reiser et al., 1985). One exception to this rule is the GUIDON system (Clancey, 1979) which tutors students in a classification task involving the diagnosis of bacterial infections. For the purposes of this research, we have elected to use a propositional Horn-clause theory refinement system and focus on categorization tasks for the following reasons. Propositional theory refinement is well understood and complete systems have already been constructed which will revise Horn-clause theories. And while categorization problems have not been a major focus of ICAI modeling efforts, concept lessons have a well understood pedagogy (Dick and Carey, 1990) and are common CAI applications. Furthermore, as Gilmore and Self (Gilmore and Self, 1988) have pointed out, machine learning has been successfully applied in categorization domains making it natural to explore its potential in concept tutorials.

2.4 Learning from Multiple Student Models

As has been stated in the previous section, theory refinement can be used to build student models and can be given preexisting bug information via modification of the initial theory. It follows, then, that any additional bugs found during modeling can be incorporated for later use in the same way. Specifically, if the system found any new trends in student behavior not previously noted, these could be added by further modifying the initial theory given to the theory refinement system. Furthermore, if such trends could be detected automatically, the system could be started with *no* preexisting knowledge except the correct domain theory, and over time a bug library would be incrementally constructed. This would not only greatly reduce the cost of building a bug library, but would also enable the modeler to tailor itself continuously using the students it encounters. The difficulty, of course, lies in automatically detecting the trends in student behavior.

To find trends in student behavior, one must obviously look at more than one student and the students must be compared on their reactions to similar inputs. Furthermore, for any trend to be meaningful to the modeling system, it must be expressed in the language used by the modeler. Since a theory refinement system models the student as a set of refinements to a correct domain theory, any common misconceptions must be expressed as common refinements. The task, then, becomes a search among multiple student models for a set of common refinements.

If multiple students are modeled using examples that exercise similar portions of the theory, the refinements used to create each student model are likely to have similarities which can be extracted. This is accomplished by feeding all of the refinements from the student models into a *secondary* learning cycle which looks for overlaps among the refinements that occur in a majority of the student models. The resulting subset of refinements is termed *stereotypical* since it represents the average changes made across the whole population of students. Such information is ideal for feeding back into the modeling process because it highlights changes which are likely to be required when constructing a model.

Theory refinement techniques, then, provide the basis for a *learning to model* paradigm. Regardless of the theory refinement algorithm used, the result is a general mechanism for dynamic student modeling and for automatic bug library construction which is only dependent upon the language used to represent student knowledge.

3 Theory Refinement Algorithm

For its student modeling component, Assert uses a propositional Horn-clause theory refinement system based upon the Either theory refinement algorithm (Ourston and Mooney, 1990; Ourston and Mooney, in press; Ourston, 1991). Either was selected because it is the most complete theory refinement system available. It can generalize or specialize a theory, and is guaranteed to produce a set of refinements which are consistent with the input examples. Unfortunately, Either's worst case run-time is exponential in the size of the theory, making it too slow for use in an interactive tutoring domain. For this reason, a modified version of Either called Neither (New Either) was developed. Neither computes refinements in time linearly proportional to the size of the theory while maintaining Either's guaranty of consistency with the input examples. Following is a description of the Either

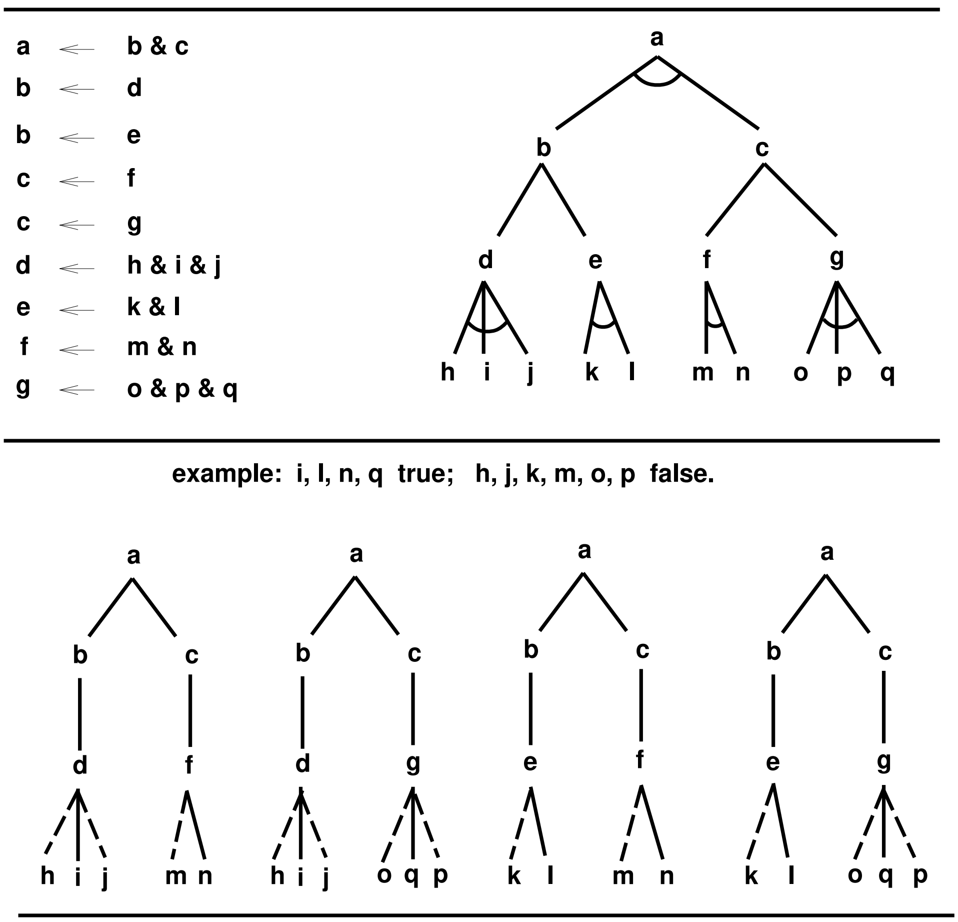

Figure 1: Partial proofs for an unprovable positive example. Unprovable antecedents are shown with dotted lines.

algorithm and the extensions made in NEITHER.

3.1 The EITHER Algorithm

EITHER was designed to repair propositional Horn-clause theories that are either overly-general or overly-specific or both using a set of input examples. The examples are assumed to be in the format of a list of values for a given set of domain features called *observable* features. Each example has an associated category which should be provable using the theory with the feature values in the example. An overly-general theory is one that causes an example (called a *failing negative*) to be classified in categories other than its own. EITHER specializes existing antecedents, adds new antecedents, and retracts rules to fix such problems. An overly-specific theory causes an example (called a *failing positive*) not to be classified in its own category. EITHER retracts and generalizes existing antecedents and learns new rules to fix these problems. Unlike other theory refinement systems that perform hill-climbing (and are therefore subject to local maxima), EITHER is guaranteed to fix any arbitrarily incorrect propositional Horn-clause theory (Ourston, 1991).

As an example of how EITHER repairs a theory, refer to Figure 1. This illustration depicts

```
Compute all minimal repairs for each example
While some examples remain uncovered
    Add best repair to covering set
    Remove examples covered by repair
end
Apply repairs in covering set to theory
```

Figure 2: Main loop of the EITHER algorithm.

a process during theory generalization where EITHER is searching for *leaf rule* antecedent retractions to correct a failing positive example. A leaf rule is a rule whose antecedents include an observable feature or an antecedent that is not the consequent of any existing rule. The upper half of the diagram shows an input theory both as rules (on the left) and as an AND-OR graph. The lower half of the diagram shows a hypothetical failing positive example and its *partial proofs*. A partial proof is one in which some antecedents cannot be satisfied. From these proofs there are four possible repairs which will fix the example: retract `h,j,m`; retract `h,j,o,p`; retract `k,m`; retract `k,o,p`. Theory specialization follows a similar process to return sets of leaf-rule retractions which fix individual failing negative examples.

The overall algorithm used by EITHER for both generalization and specialization is shown in Figure 2. There are three basic steps. First, all minimal leaf-rule repairs for each failing example are computed. Next, EITHER enters a loop to compute a small subset of these repairs that can be applied to the theory to fix all of the failing examples. This subset is called a *cover*. Repairs are ranked according to a benefit-to-cost ratio that trades off the number of examples covered against the size of the repair. The best repair is added to the cover on each iteration. Lastly, the repairs in the cover are applied to the theory. If the application of a repair over-compensates by creating new failing examples, EITHER passes the covered examples and the new failing examples to an induction component based upon a version of ID3 (Quinlan, 1986). The results of the induction are added as a new rule when generalizing or as additional antecedents when specializing.

3.2 Building NEITHER from EITHER

The main loop of NEITHER, shown in Figure 3, is quite different from EITHER (see Figure 2). Two new algorithms form the basis for the difference. First, calculation of repairs is now achieved in linear time. Second, all searches through the theory (for deduction, antecedent retraction and rule retraction) are optimized in NEITHER to operate in linear time by marking the theory to avoid redundant subproofs. NEITHER abandons the notion of searching for all partial proofs in favor of rapidly selecting a *single* best repair for each example. The three steps of the old EITHER algorithm can then be integrated into a single loop.

To illustrate how repairs are computed in linear time, refer again to Figure 1. Rather than computing all partial proofs, NEITHER follows a recursive bottom-up procedure to construct a single set of retractions. Starting at the leaves of the tree, NEITHER collects

```
While some examples remain
    Compute a single repair for each example
    Apply best repair to theory
    Remove examples fixed by repair
end
```

Figure 3: Main loop of the NEITHER algorithm.

the set of unprovable antecedents and passes them up to the parent node. At each parent, the best option from among its children is computed and passed up again. When multiple options exist at a parent node, NEITHER alternates between returning the smallest option and returning the union of the options, depending whether the choice involves an AND or OR node. For generalization, retractions are unioned at AND nodes because all unprovable antecedents must be removed to make the rule provable. At OR nodes, only the smallest set of retractions is kept since only one rule need be satisfied. For specialization, these choices are reversed. Results are unioned at OR nodes to disable all rules which fire for a faulty concept. At AND nodes, the smallest set of rule retractions is selected since any single unsatisfied antecedent will disable a rule.

As an example, in Figure 1 the antecedent retraction calculations made by NEITHER would begin at the root of the graph, recursively calling nodes `b` and `c`. Retraction for node `b` would then recurse on nodes `d` and `e`. Since `h`, `j` and `k` are false, node `d` returns `(h,j)` and node `e` returns `(k)`. When the recursion returns back to node `b` a choice must be made between the results from nodes `d` and `e` because the theory is being generalized and node `b` is an OR node. Since node `e` requires fewer retractions, its retractions are chosen as the return value for node `b`. Recursion for node `c` follows a similar pattern: node `f` returns `(m)`, node `g` returns `(o,p)` and node `c` chooses the smaller results from node `f` as its return value. Finally, nodes `b` and `c` return their values to node `a`. Now, since node `a` is an AND node and the theory is being generalized, the results from `b` and `c` are combined. The final repair returned from node `a` is retract `(k,m)`.

Note that this algorithm is linear in the size of the theory. No node is visited more than once, and the computation for choosing among potential retractions must traverse the length of each rule at most once. The final repair is also minimum with respect to the various choices made along the way; it is not possible to find a smaller repair that will satisfy the example. This new algorithm thus trades the multiple repair options available in the partial proofs for speed in computation.

3.3 A NEITHER Example

Figure 4 illustrates a more complete example of how NEITHER revises a theory to be consistent with a set of examples. The figure is separated into four parts. Part (a) shows an original theory consisting of two rules which define the concept `a`. Part (b) shows four examples given as input to NEITHER. The first three of these are failing positives; *i.e.*, the original

Original Theory		
a	$\leftarrow$	b c d e f
a	$\leftarrow$	u v w x y

(a)

Example	*Features*	*Category*
E1	$\bar{b}\ \bar{c}\ d\ e\ f\ \bar{u}\ \bar{v}\ \bar{w}\ x\ y$	a
E2	$\bar{b}\ \bar{c}\ \bar{d}\ e\ f\ \bar{u}\ \bar{v}\ \bar{w}\ \bar{x}\ y$	a
E3	$\bar{b}\ \bar{c}\ d\ \bar{e}\ f\ \bar{u}\ \bar{v}\ w\ x\ y$	a
E4	$\bar{b}\ \bar{c}\ \bar{d}\ \bar{e}\ \bar{f}\ \bar{u}\ \bar{v}\ w\ x\ y$	$\bar{a}$

(b)

	Iteration 1			*Iteration 2*		
	repair	*benefit*	*cost*	*repair*	*benefit*	*cost*
E1	(b,c)	1	2	n/a	n/a	n/a
E2	(b,c,d)	2	3	n/a	n/a	n/a
E3	(u,v)	1	3	(e)	1	1
E4	n/a	n/a	n/a	n/a	n/a	n/a

(c)

Theory after Iter. 1		
a	$\leftarrow$	e f
a	$\leftarrow$	u v w x y

Final Theory		
a	$\leftarrow$	f
a	$\leftarrow$	u v w x y

(d)

Figure 4: Example of NEITHER's refinement algorithm.

theory is unable to prove `a` true for any of these three examples. The fourth example is a negative example that the original theory correctly classifies. Though such correct examples initially pose no problem, they must be considered when selecting the best repair to apply to the theory.

Part (c) shows the effects of two iterations through the loop of Figure 3. On the first iteration, NEITHER computes a repair for each of the three failing positive examples as outlined in the previous section. Recall that a theory must be generalized to fix failing positives, and that a concept with multiple rules is defined as an OR node in the AND/OR graph representation of a theory. Thus during calculation of repairs, NEITHER will select the smaller of the fixes proposed for the two rules. For example E1, the repair `(b,c)` is smaller than `(u,v,w)`, for E2 `(b,c,d)` is smaller than `(u,v,w,x)`, and for E3 `(u,v)` is smaller than `(b,c,e)`. Since E4 is correctly classified, it requires no repair.

Having found the three repairs, NEITHER must then choose the best among them to apply to the theory before moving to the second iteration. To rank each repair, NEITHER computes a benefit-to-cost ratio by temporarily applying each repair to the theory in turn. The benefit is measured in terms of how many failing examples will be fixed by applying the repair. The cost is the sum of the size of the repair plus the number of any *new* failing examples created. Applying `(b,c)` to the original theory will only fix example E1, giving it a benefit value of 1. Since no new failures are created by this fix, the cost for it is just 2. E2's repair will fix both E1 and E2, giving it a benefit of 2 at a cost of 3 since it also avoids any new failures. E3's repair only covers E3, but does so at the expense of causing E4 to be incorrectly classified. Thus an additional penalty of one (for one new failing example) is added to the cost of E3's repair. Since E2's repair has the best benefit-to-cost ratio, it is selected and applied to the original theory.

NEITHER then enters the second iteration of its main loop. Note however that the original theory has been modified by the application of the repair from the first iteration. The new theory is shown in the top half of part (d) of Figure 4. At this point, only one example, E3, remains a problem. The repair from the last iteration fixed E1 and E2 without disrupting the classification of E4. The repair computation for E3 is now different from the last iteration due to the change made in the theory. Since `b`, `c` and `d` have been removed from the first rule, NEITHER selects `(e)` as the repair for E3 since it is smaller than `(u,v)`. Note also that this new repair for E3 will not cause E4 to fail. Since there are no other repairs, NEITHER applies this fix to the theory resulting in the final theory shown. If the four examples had represented categorizations generated by a student, and if the original theory represented the correct domain knowledge, then the difference between the original theory and the final theory would represent the student model. Specifically, the student would be modeled as missing the first four conditions of the first rule of the theory.

3.4 Comparison of NEITHER and EITHER

The DNA promoter sequences data set (Towell et al., 1990) was used as an initial test to measure the benefits of the NEITHER approach. This data set involves 57 features, 106 examples, and 2 categories. The theory provided with the data set has an initial classification accuracy of 50%. The main point of the test was to illustrate whether NEITHER could maintain the predictive accuracy of EITHER while reducing execution time. Other tests of the NEITHER algorithm are described in (Baffes and Mooney, 1993).

The experiments proceeded as follows. Each data set was divided into training and test sets. Training sets were further divided into subsets, so that the algorithms could be evaluated with varying amounts of training data. After training, each system's accuracy was recorded on the test set. To reduce statistical fluctuations, the results of this process of dividing the examples, training, and testing were averaged over 25 runs. Training time and test set accuracy were recorded for each run. Statistical significance was measured using a Student t-test for paired difference of means at the 0.05 level of confidence (i.e., 95% confidence that the differences are not due to random chance).

The results of our experiments are shown in the Figures 5 and 6. Figure 5 compares the accuracy of NEITHER and EITHER on the test set as a function of the number of training examples. NEITHER's accuracy was lower than EITHER's for small training sets and higher for large training sets. Figure 6 compares the running times for NEITHER and EITHER. NEITHER consistently ran more than an order of magnitude faster than EITHER. These timing results were not surprising since NEITHER uses a linear approach to repair calculation as opposed to EITHER's potentially exponential method. Since a repair must be computed for every failing example, faster repair calculation translates into a dramatic overall time savings.

The most surprising result of the experiment was the difference in accuracy between EITHER and NEITHER. As stated above, EITHER was more accurate with fewer training examples, but its accuracy dropped off relative to NEITHER as the number of examples increased. One possible explanation for this behavior lies in the difference between how the two systems compare potential refinements. Recall that EITHER computes multiple repairs for each example, but does so only once. NEITHER, by contrast, computes one repair per example each time through its main loop. As a result, with fewer training examples, EITHER has more potential refinements to examine, apparently giving it an edge over NEITHER.

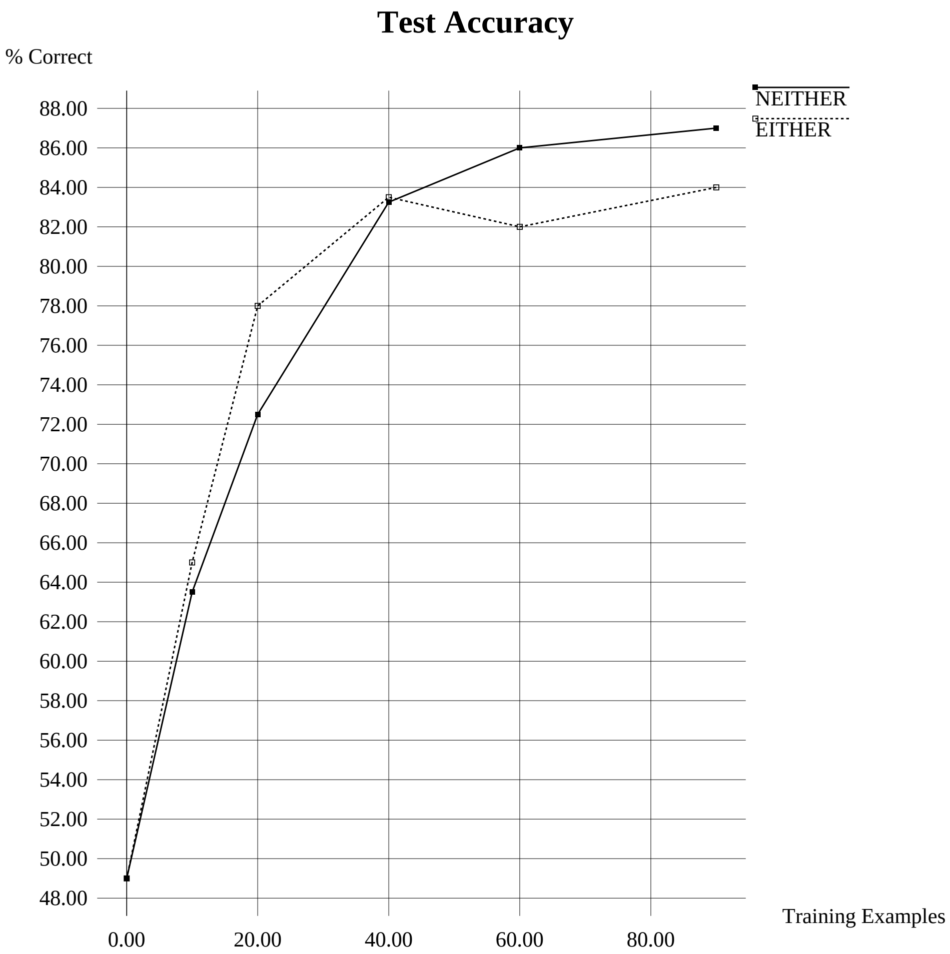

Figure 5: Accuracy of EITHER and NEITHER on DNA promoter test.

Even though NEITHER computes new repairs each time it iterates, there may not be enough iterations in some cases to generate as rich a set of repairs as is done in one step by EITHER. On the other hand, as the number of training examples grows, NEITHER undergoes many more iterations, each computing new repairs *in light of any previous refinements*. Since EITHER computes its repairs for each example independently, it can miss some interactions which might occur when the refinements are applied to the theory in a particular order. Capturing these interactions may be one reason NEITHER out-performs EITHER with large numbers of examples. In any event, NEITHER produces results very close to EITHER's in a fraction of the time, making it much more suitable for use in an interactive setting such as tutoring.

4 Finding Trends in Student Behavior

Having described how ASSERT uses NEITHER to rapidly construct student models, it remains to be seen how trends across student models can be automatically detected and then fed back into the modeling process. As discussed in Section 2.4, this problem reduces to a search

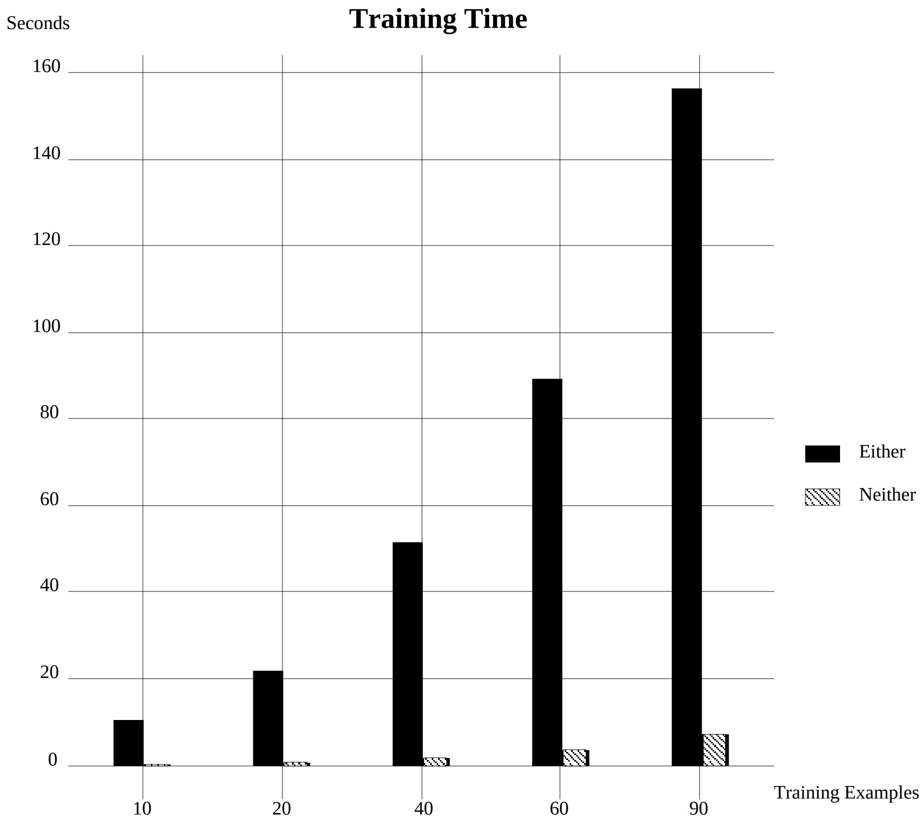

Figure 6: Run time of EITHER and NEITHER on DNA promoter test.

within the space of possible refinements to the original theory for a subset of common refinements. That is, given a measure of how each particular refinement would effect NEITHER's ability to model the average student, one can search through all refinements from the student models collecting those which would have a positive impact on the modeling process. When these common refinements are added to the original theory, the result is termed a *stereotypical theory*.

4.1 Construction of a Stereotypical Theory

Figure 7 depicts the general motivation behind the construction of a stereotypical theory as a method for incorporating the trends across students. The whole idea is tied to the distance between theories. For any two theories the distance between them can be measured in terms of the number of literal additions and deletions required to transform one theory into another. Recall that the task of a theory refinement system like NEITHER is made *simpler* when the original theory is closer to the target theory. This is because the number of literal additions or deletions which must be applied to transform the original theory is smaller. Thus, any method which moves the original theory closer to its target will have a positive effect on the theory refinement process.

The left half of Figure 7 shows an original theory and four student model theories. The separation between the circles indicates the distance between the theories; thus, SM_2 was

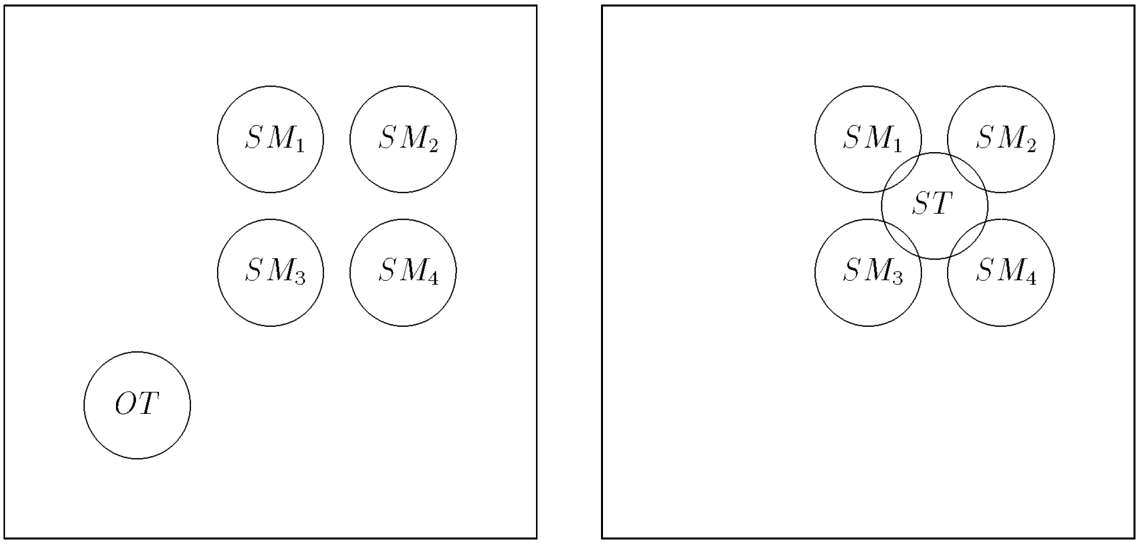

Figure 7: Diagram of stereotypical theory construction. SM stands for "student model," OT for "original theory" and ST for "stereotypical theory."

the most difficult theory to generate starting from the original theory OT. The right half of Figure 7 shows a stereotypical theory which is centered among the student models. Starting Assert from ST rather than OT decreases the effort required to model the students and increases the chances of producing an accurate model. The important point to realize from this diagram is that the *sum of the distances from ST to the student models is less than the sum from OT to the student models*. In fact, ST represents a starting theory with the minimum total distance from the student models. Consequently, the process for computing ST from the original theory and the student models amounts to minimizing the following sum:

$$\sum_{m=SM} distance(OT', m)$$

where OT' is any revised theory formed by applying some subset of all the refinements of the student models to the original theory. When this sum is minimized, $OT' = ST$, the stereotypical theory. The stereotypical theory is thus an embodiment of the trends among the student models which can be used directly by theory refinement for future modeling efforts.

Note that searching among all possible refinements to the original theory to find the one which has minimum total distance from the student models is an exponential process. Any single refinement or any combination of refinements which may or may not be present in the student models could be explored. To avoid this, Assert uses the refinements from the student models to focus a heuristic search for constructing a stereotypical theory. The result is hill-climbing search consisting of two nested loops. The inner loop searches for the best refinement to add from among all the refinements of the student models. The outer loop incrementally constructs the stereotypical theory by adding each refinement found by the inner loop in turn.

To make the algorithm for stereotypical theory construction complete, the inner loop must perform the additional task of *generalizing* among the refinements of the student models.

Generalization is important for finding commonalities among refinements which may be similar, but not identical. For example, imagine the case where three different students models have all resulted in a set of antecedent deletions from the same rule. It may be that no two of these sets are the equal, yet all share a common subset. Without the ability to extract such a subset, ASSERT would be severely limited in its ability to extract trends in student behavior.

Fortunately, a straightforward technique exists for forming generalizations among refinements. Since any refinement to a propositional theory can be expressed as a logic clause, one can use the the *least general generalization* (LGG) operator (Plotkin, 1970). When two propositional logic clauses are not identical, one can form a generalization of the two by dropping literals from the clauses. Any number of literals may be omitted, but the most specific (*i.e.*, least general) way to generalize the two clauses is to drop only those literals which appear in just one of the two clauses. This is the same thing as taking the intersection between the two clauses. Since refinements in NEITHER are collections of propositional logic literals, the LGG of two refinements is simply their intersection. Note that the result of forming the LGG of two refinements is also a refinement, which can be used in subsequent LGG operations.

Selecting the best refinement to add to the stereotypical theory becomes a two step process. First, the best potential refinement from among the student models is selected as a seed. The best refinement is the one which, if added to the current stereotypical theory, reduces its total distance to the student models the most. This seed refinement is then generalized against each of the other refinements using the LGG operator. As long as one of the resulting generalizations further decreases the total distance of the stereotypical theory, the generalization process continues. When no generalization will improve the refinement, it is returned and checked for addition to the stereotypical theory. The whole process halts when no generalization of any remaining refinement will reduce the total distance of the stereotypical theory to the student models. Pseudo code for stereotypical theory construction is shown in Figure 8.

4.2 An Example Stereotypical Theory

An example of how a stereotypical theory is constructed is shown in Figure 9. The top half of the diagram shows three student models and their corresponding refinements. For illustration purposes, the original theory is shown to have only a single rule. Both the first and second student models consist of a repair which deletes antecedents from the rule. Student model 1 deletes `(b,c,d,e)` and student model 2 deletes `(c,d,f)`. The third student model replaces `c` with `g`, which is the same as performing a deletion followed by an addition. Each of the student models is shown with the theory that results when the refinement is applied, and the refinements are also listed separately, numbered R1, R2 and R3.

The bottom half of Figure 9 illustrates the nine steps ASSERT follows to compute the stereotypical theory. The numeric values in the table indicate the distances between the theory shown in the leftmost column and the three student models. The rightmost column is the sum of the distances to the three student models. As described in the last section, ASSERT begins by setting ST to the original theory. Line 1 of the figure shows the total distance between this theory and the student models to be 9. The distance to student model

```
Let  R = set of all refinements from all student models
     M = list of all student model theories
     ST = current stereotypical theory
     H = current hypothesis for updating ST

initialize ST = correct theory
repeat /* outer loop */
    sort r in R by biggest decrease to ST's total distance
    for all r in R in sorted order /* inner loop */
        set H = r
        repeat while total distance using H still decreasing
            let L = LGG's of H with all elements of R
            if best of L yields lower total distance than H
                H = best of L
        end-repeat
        if H decreases the total distance of ST
            quit inner loop early
    end-for /* inner loop */
    if H decreases the total distance of ST
        update ST by applying H
    else quit outer loop
end-repeat /* outer loop */
```

Figure 8: Pseudo-code for stereotypical theory construction.

1 equals 4 since `b`, `c`, `d` and `e` must be removed to convert the original theory to this model. Only 3 antecedents must be removed to create student model 2 making its distance from the original theory 3. Finally, since `c` must be removed and `g` added to create the third model, its distance is 2 yielding a total of 9 as the distance from all the models.

Lines 2 through 4 show how the various refinements from the student models are ranked. A total distance for each is computed by temporarily applying the refinement separately to ST and computing the resulting distance to the student models. Each refinement is shown with the theory that results from applying that refinement to ST. All of the refinements decrease the total distance, but since R2 has the best overall effect it is chosen as an initial candidate to be added to ST.

Assert now enters its inner loop for determining if any generalizations can be made to R2 to further decrease the total distance. Lines 5 and 6 show the results from computing the LGG of R2 with each of the other two refinements. The LGG of R2 and R1 is `(c,d)` since that is the largest subset of deletions common to both R2 and R1. When `c` and `d` are temporarily deleted from ST, the resulting theory has a total distance of 5 from the three student models, which is an improvement over using R2 alone. In line 6, the LGG of R2 and R3 is formed by splitting the refinement R3 into a deletion followed by an addition. Since

Original Theory
a $\leftarrow$ b c d e f

Student Model 1
delete(b,c,d,e)
a $\leftarrow$ f

Student Model 2
delete(c,d,f)
a $\leftarrow$ b e

Student Model 3
delete(c), add(g)
a $\leftarrow$ b d e f g

Refinements
R1: delete(b,c,d,e)
R2: delete(c,d,f)
R3: delete(c), add(g)

	Model 1 a $\leftarrow$ f	*Model 2* a $\leftarrow$ b e	*Model 3* a $\leftarrow$ b d e f g	*Total Dist.*
1. ST: a $\leftarrow$ b c d e f	4	3	2	9
2. ST+R1: a $\leftarrow$ f	0	3	4	7
3. ST+R2: a $\leftarrow$ b e	3	0	3	6
4. ST+R3: a $\leftarrow$ b d e f g	4	3	0	7

R2 produces greatest savings. Current hypothesis: R2

5. ST+LGG(R2,R1): a $\leftarrow$ b e f	2	1	2	5
6. ST+LGG(R2,R3): a $\leftarrow$ b d e f	3	2	1	6

LGG(R2,R1) a further improvement. Hypothesis updated to: LGG(R2,R1)

7. ST+LGG(LGG(R2,R1),R3): a $\leftarrow$ b d e f	3	2	1	6

No Improvement. Update ST with LGG(R2,R1) to: a $\leftarrow$ b e f

8. ST: a $\leftarrow$ b e f	2	1	2	5
9. ST+R3: a $\leftarrow$ b e f g	3	2	1	6

No improvement. Final Stereotypical theory: a $\leftarrow$ b e f

Figure 9: Construction of a stereotypical theory. ST represents the stereotypical theory and LGG stands for “least general generalization.”

`c` is deleted in both refinements, it becomes the LGG of R2 and R3. Omitting `c` from ST moves the theory closer to model 3, but farther from model 2 resulting in no overall benefit. Since the LGG of R2 and R1 offers the best results so far, it becomes the new hypothesis after line 6.

With the hypothesis improving from generalization, the search through more LGG operations continues. Now, however, there is only one refinement left which has not already been included in the current hypothesis. Thus in line 7, an LGG is formed with the current hypothesis and R3. As this does not reduce the total distance any further, ASSERT stops searching for a better generalization and checks to see if the current hypothesis will improve ST. Since ST has not yet been permanently modified, its total distance is still 9 (line 1). With the current hypothesis, the total distance is 5 (line 5). ASSERT thus permanently modifies the ST with the hypothesis and repeats its outer loop to check if any remaining refinements can improve the stereotypical theory further. Only R3 remains, which again fails to reduce the total distance (lines 8 and 9). With no further changes possible, ASSERT exits yielding the final stereotypical theory: `a` $\leftarrow$ `b e f`.

5 Preliminary Evaluation of ASSERT

The two claims made at the end of Section 1 were that the theory refinement component of ASSERT would prove to be a more effective modeling algorithm than induction, and that using the stereotypical theory would result in better subsequent modeling. Two experiments were designed to test these claims. Due to the difficulties associated with collecting large amounts of data from real students, our first experiment used simulated student responses to generate enough data for a complete test of ASSERT. The second used a small set of real student data collect using a CAI system to teach concepts to nursing students. The primary goal of this latter experiment was to show a limited illustration of ASSERT's capabilities in a real world setting.

5.1 Animal Classification Experiment

Any simulation built to test all aspects of the ASSERT algorithm requires several tools. First, one needs the ability to simulate student responses in some reasonable fashion. These should be generated in a format which can be input to a theory refinement learner. This experiment used a multiple choice question format, which maps directly to the types of examples required by theory refinement. The elements of the question act as the feature value inputs from the domain, and the various answers serve as labels which the student can use to categorize the example.

Second, an input theory is needed which describes the correct mapping from features to categories. Furthermore, this theory should be complex enough to make the modeling problem challenging. We chose a propositional theory which classifies descriptions of animals into one of twelve categories using a set of 14 features. The full animal theory shown in appendix A is an extension of a set of rules given in (Winston and Horn, 1989). It contains four disjunctive subconcepts (bird, mammal, ungulate and carnivore) and many of the examples require a chain of multiple subconcepts to be correctly classified. The animal domain thus represents a nice theory for testing ASSERT's ability to catch misconceptions.

Third, both common and student-specific errors must be simulated. Student-specific errors should be unique to individual students. Common errors should represent recurrent trends in a population of students which ASSERT must separate from the unique errors when constructing its stereotypical theory. Predictive accuracy can then be used to measure how well ASSERT models each individual student, and at the same time to test whether or not a stereotypical theory makes any difference in the accuracy.

Our tests were run from a pool of 180 examples randomly generated using the correct animal classification rules (15 examples for each of the 12 categories). These examples simulated the "multiple choice questions" presented to the students. Artificial students were generated by making modifications to the correct theory. As each student theory was formed, it was used to relabel the 180 examples to simulate the behavior of that student. These relabeled examples acted as "answers" the student would generate to the 180 multiple choice questions.

Modifications made to the correct theory to create students were of two types. One set of modifications was predefined, with a given probability of occurrence. These simulated common errors that recurred in the student population. We used four common deviations, each with a 0.75 probability of occurrence. Two of these deleted antecedents from rules, one added an antecedent, and one changed an antecedent. To simulate individual student differences, each student theory was further subjected to random antecedent modifications and rule additions and deletions, each with a probability of 0.10.

5.1.1 Experimental Method

ASSERT was tested against both normal theory refinement and induction using a two-phased approach as first described in (Baffes and Mooney, 1992). The first phase was used to build a stereotypical theory for use in the second phase as follows:

1. First, 20 artificial students were created using the methods described above.
2. For each student, all 180 examples were relabeled using the student's theory.
3. From these 20 students, 20 student models were generated using ASSERT on all 180 relabeled examples.
4. A stereotypical theory was then built from the 20 student models using the algorithm from Section 4.1.

Three of the four common predefined bugs ended up in the stereotypical theory. The fourth was more difficult for ASSERT to generate, since it required a deletion of an antecedent followed by an addition of a different antecedent. This fourth bug ended up as two different rules in the stereotypical theory.

For the second phase, additional artificial students were generated to test ASSERT using various initial theories. A series of experiments were run starting ASSERT with (1) the correct animal rules, (2) the stereotypical theory from phase 1 above, or (3) no initial theory. With no initial theory, ASSERT defaults to inductive learning alone using a propositional version of the FOIL (Quinlan, 1990) algorithm. This phase ran as follows:

Initial Rules	*Accuracy*
stereotypical	93%
correct	86%
stereotypical, no refinement	81%
correct, no refinement	71%
none (induction)	52%

Table 1: Accuracy of student models for animal domain tests.

1. First, 10 new artificial students were generated using the same techniques used in phase 1. For each, the 180 examples were relabeled using the student's buggy theory.
2. 50 examples were randomly chosen from the 180 relabeled by the student as training examples. Each new student was modeled using ASSERT with the same 50 examples and one of the three initial theories described above.
3. The other 130 examples were reserved for testing the accuracy of each student model as follows. Recall that the output of ASSERT is a revised theory representing the student model. This theory was used to label each of the 130 test examples. These labels were compared to those generated using the student's buggy theory from step 1 to compute a percentage accuracy.

5.1.2 Results

Table 1 compares the average accuracy of ASSERT started with each of the three different initial theories. For comparison purposes, we also measured the accuracy of both the correct and stereotypical theories *without* any refinement. Statistical significance was measured using a Student t-test for paired difference of means at the 0.05 level of confidence (*i.e.*, 95% confidence that the differences were not due to random chance). All the differences shown in Table 1 are statistically significant.

Both of our hypotheses were borne out by the results presented in Table 1. First, it is apparent that theory refinement is superior to induction in terms of accuracy. This is not surprising since induction must model *correct* as well as buggy student behavior, whereas theory refinement need only alter correct rules to capture the misconceptions. The difference is even more pronounced when theory refinement proceeds from the stereotypical model. Induction simply has more work to do.

Second, our results show that theory refinement models students more accurately when given an initial rule base that approximates typical student errors. Since all the students were generated using the same criteria, providing ASSERT with the stereotypical rules effectively gives it a head start over the correct theory.

Running the correct and stereotypical theories without refinement also produced interesting results. Both outperformed induction, further illustrating the disadvantage of trying to model students from scratch using a small number of examples. It is also apparent that

NURSING PROGRESS NOTES, XYZ HOSPITAL

Date/Time	T	P	R	B/P	Nursing Observations, Page 1 of 1
11/18 1945	97.4	144	22	102/70	24 y/o male admitted to ER#4 following MVA. Pulse weak and thready; respirations shallow. C/O pain (L) shoulder and (L) lateral rib cage. Skin cool and clammy. UA send; hemastick +. IV of LR @ 75 cc/hr begun in (L) wrist with 18G angiocath. O_2 @ 2 L/min per nasal cannula applied.

Table 2: Example of shock problem showing a patient suffering from cardiogenic shock.

refinement is essential to effective modeling, even if the initial rules represent stereotypical student behavior.

Finally, it should be noted that the execution times for all the tests were below 4 seconds running under Lucid Common Lisp on a SPARC Station 2. While the code has not yet been fully optimized for speed, this is still within the tolerance required for interactive tutoring.

5.2 Nursing Student Experiment

A second set of experiments was run to test Assert in a real world domain. The data for this experiment is borrowed from a separate research project that tested the ability of nursing students to retain concepts for determining if a patient was suffering from shock (Murphy and Davidson, 1991). Data from 26 of the students is used here. Each student interacted with a CAI system which presented patient case histories and asked the student to diagnose which of three kinds of shock, if any, the patient was experiencing. A typical example of the problems seen by the student is shown in Table 2. Each example is *relatively different* from the others; specifically, there is very little overlap in the rules used to correctly classify any two examples. Also, an average of *only half of the examples were presented to each student*, drawn randomly from a pool of the 56 possible cases. A theory for diagnosing shock was written using definitions presented in the CAI system, material from among the 56 case histories and consultations with a medical expert. The final theory, shown in appendix B, has 58 rules for categorizing the three types of shock.

5.2.1 Experimental Method

In the following experiments, students were divided into two groups. The first group was used strictly for constructing a stereotypical theory; no accuracy measurements were taken. Each student of this group was modeled by Assert using all of his or her interactions with the system. The models were collected and used to construct a stereotypical theory. The second test group was used to compare the accuracy of the stereotypical and correct theories. Due to the small number of students, a leave-one-out, cross-validation method was followed where all but one of the students were used to form the stereotypical theory with

Initial Rules	*Accuracy*
stereotypical, no refinement	72%
correct, no refinement	71%
stereotypical	68%
correct	67%
induction	49%

Table 3: Accuracy of student models for nursing domain tests.

the remaining student used to test accuracy. Each student was separated from the rest and the results averaged.

Two experiments were run to test accuracy on the remaining student. First, the accuracy of the stereotypical theory and the correct theory were compared without performing theory refinement. This would illustrate whether the stereotypical theory alone had a positive effect on accuracy. In a second test, theory refinement was tested using another leave-one-out technique to model the remaining student and test the results. Each example seen by the student was held out in turn, with the rest used to form a model. The accuracy of the resulting model was tested on the remaining example and the results averaged. As a benchmark for comparison to the theory refinement, induction was also run using the same leave-one-out test.

5.2.2 Results

Despite the fact that only half of the possible cases were presented to each student and that each example was relatively different, ASSERT was still able to find enough commonality to construct a stereotypical theory. This consisted of one refinement: the rule

ineffective-pumping-action $\leftarrow$ (diagnosis MVA) (chest-pain true)

was deleted from the original theory. Out of the 26 student models, 16 (62%) had this change. Two other refinements were found to exist in 10 of the 26 student models. It seems likely, then, that had more data been collected for each student, more stereotypical refinements would have been detected.

Table 3 shows the results of both experiments. Note that all differences are significant. Several interesting observations can be made from this table. First, it is apparent that the stereotypical theory proved slightly more effective than the correct theory, both with and without the use of theory refinement to form the student models. The fact that this difference is so small in both experiments is due to both the limited amount of data collected for each student and the relative independence of the examples. Since the stereotypical refinement existed in just over half the students, and since few examples would illustrate its use, it follows that there is only a slight advantage to using the stereotypical theory. This is borne out by the modest increase in accuracy.

Second, it is apparent that theory refinement performed more poorly than simply using the correct or stereotypical theory without revision. Again, this comes from the lack of data collected for each student. In this case, the problem arises because it is not possible to generate training and test sets which can illustrate a *transfer* from the learning phase to the testing phase of the experiment. In general, improvements from a theory refinement process can best be illustrated if the training set is representative of the examples which will be found in the test set. Then, refinements made during training are more likely to result in accurate classification of the test examples. Here, however, since each example is relatively different from the others, the training and test sets are fairly disparate making it hard to get transfer between the two.

Finally, the last entry of Table 3 shows that, as with the animal experiment, induction cannot perform as well as theory refinement. Induction is better than random guessing, which has only a 25% chance of success, but falls far behind theory refinement. Again this is because induction from scratch is simply a harder task.

6 Related Work

There are three other systems which utilize machine learning techniques for student modeling. The first of these is VanLehn's SIERRA system (VanLehn, 1983) which builds on ideas developed in the DEBUGGY system (Burton, 1982). In DEBUGGY, student models are formed by introducing incorrect subskills into a lattice of skills called a *procedural network*. When a subskill is missing or incorrect, the result is faulty behavior. DEBUGGY attempts to model incorrect behavior by searching for faulty subskills via a generate-and-test method using known deviations for the subskills. VanLehn built on these ideas to develop a theory of how students form misconceptions. He proposes that faulty behavior arises from a bad *core procedure* which the student is following. When the student's procedure fails to work in a given problem, the result is an *impasse* which the student overcomes by constructing a local *REPAIR*. Incorrect core procedures are assumed to arise from incorrect inductions which occur when the examples given to the student do not meet a set of *felicity conditions* to ensure proper induction. *STEP theory* outlines the conditions for instruction which will lead to proper induction on the part of the student. While SIERRA is perhaps the most complete theory to date on how student misconceptions arise, it is intended as a cognitive model of how bugs are formed, and is thus a simulation of how the *student* forms misconceptions. ASSERT, by contrast, is a simulation of how the *teacher* learns to model misconceptions.

Sleeman describes an extension to his PIXIE system, which models arithmetic errors, called INFER* (Sleeman et al., 1990) INFER* is an extension to the bug library approach to student modeling. When a student exhibits a problem that cannot be modeled using the bugs in the library, INFER* uses the rules it has to work forward from the problem statement and backward from the solution as far as it can. The remaining *gap* is filled by inferring a new buggy rule (called *mal-rules*). The result is a data-driven method for extending the bug library which is guided by the known misconceptions. Like INFER*, ASSERT's modeling algorithm can be biased with known misconceptions, and its bug library can be extended using specific student examples. Unlike INFER*, ASSERT can also operate effectively with no prior knowledge of misconceptions and contains an algorithm for extracting common elements from multiple student models.

Langley *et al.* (Langley et al., 1984; Ohlsson and Langley, 1985) describe the ACM system which uses a domain independent induction algorithm to induce control knowledge for selecting operators to perform addition. The goal is to induce a set of control rules that will generate an operator sequence that produces the same solutions as the student. Each run of ACM starts without knowledge of when operators should be applied and induces the conditions for applying the operator from examples of student problem solving. A path connecting the problem specification to the student's solution is found, and induction is then performed by noting whether each operator lies on or off the solution path. Since ACM starts from scratch, it must spend time modeling both correct and buggy student control knowledge that could be preprogrammed. Also, there is no facility within ACM for building in typical student bugs nor for generalizing across students to improve modeling over time.

As was discussed in Section 2.3, ASSERT further differs from all of the above systems in that it operates in a concept learning domain rather than a procedural one. While such categorization problems have not been a major focus of ICAI modeling efforts, concept lessons are well understood and are common CAI applications. Other tasks, specifically procedural ones, in general cannot be represented with the propositional Horn-clauses used by NEITHER. However, it is important to point out that the basic technique of using theory refinement for student modeling is not limited to categorization domains since other theory refinement algorithms may use different underlying representations. It is also important to reinforce a point first raised by Clancey's work on the GUIDON system (Clancey, 1979). Like GUIDON, the models build by ASSERT will only be as good as the underlying rule base from which they are drawn. A sufficient level of detail must be provided in the initial theory for ASSERT to generate a useful student model.

7 Future Work

There are several problems with the current implementation of ASSERT and the methods used for its evaluation. NEITHER should be enhanced to include other capabilities present in EITHER, and the use of stereotypical refinements should be more closely integrated with the refinement process. Also, a more complete test using real student data should be run which includes a method of remediation that illustrates the utility of stereotypical refinements. Each of these topics is addressed below in turn.

7.1 Enhancing NEITHER

The NEITHER algorithm, upon which ASSERT depends, contains two biases in its current form which limit ASSERT's modeling potential. Recall that NEITHER constructs refinements from the bottom up, keeping only the best as the search proceeds. This means that NEITHER is biased towards changes at the leaf rules of a theory, and will only look for changes in higher concepts if all leaf rule possibilities have been exhausted. Also, when forced to turn to induction, NEITHER can only learn rules which use the observable features in a domain. It is unable, for example, to induce a rule that makes use of an intermediate concept that appears in the theory. These two facts would limit any remediation techniques to the fringe concepts of a theory.

To solve these problems, two capabilities from the original EITHER algorithm will be added to NEITHER. First, a *constructive induction* technique will be introduced to allow the induction of rules with intermediate concepts. As each example is passed to induction, its set of feature values will be extended to include any intermediate subconcepts which the example proves. Adding these subconcepts as features of the example allows for their use without changing the induction algorithm.

Second, NEITHER will be modified to attempt higher level concept changes as part of its refinement detection process. This can be done by expanding the number of candidate refinements considered at each rule of the theory. Rather than having the parent nodes in the AND/OR graph simply choose from among the refinements passed up by its children, any additional refinements which could take place at the parent will also be considered. These will be ranked according to the same benefit-to-cost ratio as the child refinements, and the best will be selected. This will allow ASSERT to propose refinements at any level in the theory.

7.2 Using Stereotypical Bugs to Find Repairs

Building a stereotypical theory is only one method for incorporating common student misconceptions into ASSERT and may not be the most effective. For example, if a student happens to have *no* misconceptions, ASSERT will actually have a *harder* time modeling the student using a stereotypical theory. This is because the refinements are hard wired as part of the input; there is no option to avoid using the common bugs. Rather than changing the input theory, ASSERT will be modified to consider the common bugs as part of the process for detecting refinements. This would allow ASSERT to selectively choose from among the stereotypical bugs, disregarding any which do not apply to the current student.

One method for doing this is to incorporate the refinements of the student models directly into the repair computation process used by NEITHER to fix a failing example. Rather than modifying the input theory permanently, the idea is to annotate the theory with the refinements from the student models, biasing the repair calculation to look at the previous refinements. There are two pieces of information inherent in ASSERT's stereotypical theory algorithm (see Figure 8) which can be used to annotate the theory. These are the location where the refinement is made and the amount by which the refinement decreases the total distance to the student models.

With these two facts the theory can be annotated as follows. Each refinement from the student models can be listed as a potential change at the point in the theory it effects. If multiple refinements effect the same rule, they can be listed in order of greatest decrease to the total distance. When NEITHER then computes a repair for a future student, the refinements from the student models can be tried in addition to the normal repairs calculated. Previous refinements will be given preference over new refinements. If two or more previous refinements apply, then the one with the greatest reduction to total distance will be preferred. If no previous refinements apply, then new refinements which use components of the previous refinements will be preferred. As the search space grows, various heuristics can be used to keep the search tractable, such as using a beam search or keeping only those refinements which generated an actual savings in total distance. The result should be a more effective modeling algorithm since it will allow ASSERT to make use of any part of the refinements detected in previous students.

7.3 Adding Remediation Techniques to ASSERT

Evaluating ASSERT by measuring the predictive accuracy of its student models may not be the most appropriate appraisal. It is possible for ASSERT to construct a model which has poor accuracy and yet is useful for correcting student errors. For example, imagine a case where several student models delete the same rule of the input theory, but each replaces it with different result. A stereotypical theory in such a case would only catch the rule deletion since none of the additions are the same. The resulting theory would not be guaranteed to be any better since the rule deletion only partially corrects each student's problem. Yet, knowing the rule which was deleted could allow ASSERT to provide useful instruction or *remediation* to the student. Consequently, to thoroughly test the utility of the student model, one must perform some kind of remediation and test its effect upon student performance.

Two types of remediation will be added to ASSERT. The first will be a form of direct communication of the refinements used to construct the student model. The motivation for this type of remediation is to explicitly show the student how his or her misconception fits into the context of the overall correct domain theory. Explanations will be generated using generic templates triggered by the types of refinements made to each rule. Each type of refinement will have its own template, with the specifics filled in using the antecedents and consequent of the rule. For example, if conditions are removed from a rule, a standard explanation will be generated noting the missing conditions and repeating that they are essential for drawing the particular conclusion of the rule. To provide context, this explanation process will be recursively invoked down the chain of inferences made for the final correct categorization of the example.

The second type of remediation will be a presentation of examples that illustrate the student's misconception. These will be selected from a pool of examples covering all aspects of the domain theory. The refinement being explained will be temporarily added to the correct theory, and any example which is incorrectly classified will be added to the list of remediation examples. Again to provide context, each example will be presented along with a trace of how the erroneous category is concluded. While fairly primitive by current explanation standards, these two forms of remediation are a relatively simple means of providing a thorough accounting of the information available in the student model.

7.4 A More Thorough Test of ASSERT

While an extensive classroom-based test of ASSERT as outlined in (Legree and Gillis, 1991) is beyond the scope of this research, we hope to demonstrate an increase in student performance when using ASSERT with the remediation techniques outlined in the last section. Our experiments will use the domain of teaching type definition and declaration concepts in the C++ language. I chose this domain for two reasons. First, my personal experience as a teaching assistant for C++ continuing education classes has revealed certain areas which are repeatedly difficult for students to learn. The problems occur with type constraint concepts. Since these are definitional in nature and may be represented as a classification problem using a propositional theory, it should be possible for ASSERT to detect these problem areas as stereotypical revisions. Second, due to the current popularity of the subject, many students are readily available. This, along with the fact that I will have direct control over the data collected for each student should make it easy to collect enough data to avoid the problems found with the nursing students.

We will follow the three-part testing method exemplified by the series of experiments on the PIXIE system (Sleeman et al., 1987) with a slight modification. Students will be divided into three groups, with each receiving different feedback. All three groups will receive instruction from a standard lecture format (not a ICAI system), and then each will be given a set of test questions. The first group will be considered a control group and will be given no feedback. The primary purpose of this group is to measure the accuracy of ASSERT's modeling algorithm. The second group will be given feedback in the form of reteaching, which is the standard CAI approach. Reteaching amounts to showing the student his or her errors followed by a recap of all the material which was presented. This group will allow us to compare the remediation techniques of ASSERT against the standard CAI approach. The third group will be modeled using ASSERT with any stereotypical information gleaned from models formed (but not used) with the first two groups. Remediation will be given as outlined in Section 7.3.

A final post test will be given to each student to evaluate ASSERT's performance and compare the three remediation strategies. The models formed for each student can be used to directly test ASSERT's modeling capabilities by using the model to predict the answers to the post test and comparing the results to the actual answers generated by the student. Average performance on the post test will be used to compare the efficacy of the three types of remediation. Given class sizes of 10 to 30 students, we hope to be able to present evidence in support of the techniques used by ASSERT. Specifically, we hope to show that both the ASSERT and CAI groups perform better than the control group, and that the remediation techniques in ASSERT are more effective than reteaching CAI approach.

8 Conclusions

A new student modeling system called ASSERT has been described. ASSERT uses domain independent learning algorithms to model unique student errors and to automatically construct bug libraries. ASSERT consists of two learning phases. The first is an application of theory refinement techniques for constructing student models from a correct theory of the domain being tutored. The second learning cycle automatically constructs the bug library by extracting common refinements from multiple student models which are then used to bias future modeling efforts. Initial experimental data suggests that ASSERT is a more effective modeling system than other induction techniques previously explored, and that the automatic bug library construction significantly enhances subsequent modeling efforts. Future enhancements to ASSERT include extending it to include capabilities present in EITHER and integrating the use of stereotypical refinements more closely with the theory refinement process. Finally, a more complete test using real student data will be run to illustrate the utility of stereotypical refinements.

9 Acknowledgments

Many of the ideas presented here were developed during discussions with my research advisor Dr. Ray Mooney. I would also like to thank Dr. Marylin Murphy for generously donating the nursing student data. Thanks also to Dr. Thomas Baffes for his invaluable help in constructing the shock theory and to Chris Whatley for his help in implementing Neither. Finally, thanks to John Zelle, James Lester, Ken Samuel and Cheng-Chih Wu of the ITS group for many helpful, fruitful, discussions.

A Animal Classification Theory

Domain Features:

feed-young:	(milk regurgitate bring none)
body-covering:	(scales hair feathers moist-skin)
birth:	(live egg)
eat-meat:	(true false)
fly:	(true false)
teeth:	(pointed flat none)
fore-appendage:	(wing leg fin)
foot-type:	(hoof clawed webbed none)
neck-length:	(short medium long extra-long)
body-length:	(small medium large huge)
color:	(white black gray tawny)
pattern:	(spots stripes patch none)
pattern-color:	(black white none)
ruminate:	(true false)

Theory:

bird	← (body-covering feathers)
	← (birth egg) fly
mammal	← (body-covering hair)
	← (feed-young milk)
	← (birth live)
ungulate	← mammal (foot-type hoof)
	← mammal ruminate
carnivore	← eat-meat
	← (teeth pointed) (foot-type clawed)
bat	← mammal (color black) (pattern none) fly (pattern-color none)
duck	← bird (foot-type webbed) fly
zebra	← ungulate (color white) (pattern stripes) (pattern-color black)
tiger	← mammal carnivore (color tawny) (pattern stripes) (pattern-color black)
whale	← mammal (fore-appendage fin) (color gray) (body-covering moist-skin) (body-length huge)

giraffe	←	ungulate (neck-length extra-long) (color tawny) (pattern spots) (pattern-color black)
cheetah	←	mammal carnivore (color tawny) (pattern spots) (pattern-color black)
dolphin	←	mammal (fore-appendage fin) (color gray) (body-covering moist-skin) (body-length medium)
ostrich	←	bird (not fly) (neck-length medium) (color white) (pattern patch) (pattern-color black)
penguin	←	bird (not fly) (color white) (pattern patch) (pattern-color black) (foot-type webbed)
grackle	←	bird (color black) (pattern none) fly (pattern-color none)
platypus	←	mammal (birth egg) (foot-type webbed)

B Shock Theory

Domain Features:

temp-value:	(below-99 above-99)
pulse-value:	(above-100 above-120 below-100)
pulse-delta:	(rising falling stable)
resp-value:	(above-20 above-30 below-20)
resp-delta:	(rising falling stable)
blood-pressure-value:	(below-90 above-90)
blood-pressure-delta:	(falling rising stable)
urine-output:	(high low normal)
mental-status:	(strained normal)
skin:	(cool-clammy hot-flushed pale normal)
age:	(infant child adult elderly)
history:	(HTN cerebral-palsy maternal diabetic cardiac liver hyperthyroidism liver depression rheumatic-fever)
diagnosis:	(surgery bleeding open-wound head-wound MVA diarrhea burns hematemesis UTI bowel leukemia bee cardiac edema hyperthyroidism dysuria NSVD)
breathing:	(normal rales missing shallow wheeze)
sinus-rhythm :	(nsr rare pvc pac fibrulation)
pin-prick:	(true false)
girth:	(increasing normal)
edema:	(true false)
jaundice:	(true false)
wound-healing:	(normal drainage not-healing)
injury-time:	(recent hours days)
ngt-green:	(true false)
foul-odor:	(true false)
vomit:	(mild extensive)
diarrhea:	(mild extensive)

blood-loss:	(small large)
hemastick:	(positive negative)
guiac:	(positive negative)
hematocrit:	(normal dropping)
bowel:	(active quiet)
pedal-pulse:	(weak strong)
fetal-pulse-delta:	(rising falling stable none)
fetal-bp-delta:	(rising falling stable none)
allergen:	(bee tetanus-shot tampons pyelogram anesthesia)
allergy-symptoms:	(true false)
sputum:	(yellow red normal)
chest-pain:	(true false)
chills:	(true false)
injury-location:	(head arm leg chest abdomen torso general)

Theory:

hypovolemic	← shock disrupted-blood-volume
cardiogenic	← shock ineffective-pumping-action
vascular-tone	← shock disrupted-vascular-tone
shock	← pulse-rapid respiration-high bp-low
	← pulse-rapid respiration-high skin-abnormal
	← pulse-rapid respiration-high (urine-output low) (mental-status strained)
	← (fetal-pulse-delta falling) (fetal-bp-delta falling) (blood-loss large)
pulse-rapid	← (pulse-value above-100) not-newborn
	← (pulse-value above-120)
	← (pulse-delta rising)
bp-low	← (blood-pressure-value below-90)
	← (blood-pressure-delta falling)
respiration-high	← (resp-value above-20) not-newborn
	← (resp-value above-30)
	← (resp-delta rising)
not-newborn	← (age child)
	← (age adult)
	← (age elderly)
skin-abnormal	← (skin cool-clammy)
	← (skin hot-flushed)
disrupted-blood-volume	
	← symptoms-blood-loss
	← symptoms-fluid-loss
symptoms-blood-loss	
	← (blood-loss large)
	← (blood-loss small) (injury-time hours) infection-not-indicated
	← (hemastick positive)
	← (hematocrit dropping)
	← (guiac positive) (diagnosis hematemesis)

← (girth increasing) (wound-healing drainage)

symptoms-fluid-loss
← (vomit extensive) infection-not-indicated
← (diarrhea extensive) infection-not-indicated
← (diagnosis burns) (injury-time recent)

ineffective-pumping-action
← (diagnosis MVA) (chest-pain true)
← symptoms-cardiac
← symptoms-cardiac (history cardiac)
← (diagnosis cardiac)

symptoms-cardiac
← (sinus-rhythm pac)
← (sinus-rhythm pvc)
← (sinus-rhythm fibrulation)
← (sputum red)

infection-not-indicated
← (allergy-symptoms false) (foul-odor false) (sputum normal) (ngt-green false)
← (allergy-symptoms false) (foul-odor false) (sputum normal) (ngt-green unknown)
← (allergy-symptoms false) (temp-value below-99)

disrupted-vascular-tone
← symptoms-infection (temp-value above-99)
← allergen-present (allergy-symptoms true)
← symptoms-neural

symptoms-infection
← (ngt-green true) (bowel quiet)
← (diagnosis surgery) (injury-location abdomen)
← (diagnosis leukemia)
← (sputum yellow)
← (foul-odor true)
← (diagnosis UTI)

allergen-present ← (allergen tampons)
← (allergen tetanus-shot)
← (allergen bee)
← (allergen pyelogram)

symptoms-neural ← (diagnosis MVA) (injury-location head)
← (allergen anesthesia)

References

Anderson, J. R., Boyle, C. F., and Reiser, B. J. (1985). The geometry tutor. In *Proceedings of the Ninth International Joint conference on Artificial intelligence*, pages 1–7. Los

Angeles, CA.

Baffes, P. and Mooney, R. (1993). Symbolic revision of theories with M-of-N rules. In *Proceedings of the Thirteenth International Joint Conference on Artificial intelligence*. Chambery, France.

Baffes, P. and Mooney, R. J. (1992). Using theory revision to model students and acquire stereotypical errors. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, pages 617–622. Bloomington, IN.

Brown, J. S. and Burton, R. R. (1978). Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive Science*, 2:155–192.

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

Burton, R. R. (1982). Diagnosing bugs in a simple procedural skill. In Sleeman, D. H. and Brown, J. S., editors, *Intelligent Tutoring Systems*, chapter 8. London: Academic Press.

Carbonell, J. R. (1970). Mixed-initiative man-computer instructional dialogues. Technical Report BBN Report No. 1971, Cambridge, MA: Bolt Beranek and Newman, Inc.

Carr, B. and Goldstein, I. (1977). Overlays: a theory of modeling for computer-aided instruction. Technical Report A. I. Memo 406, Cambridge, MA: MIT.

Clancey, W. J. (1979). *Transfer of Rule-Based Expertise through a Tutorial Dialogue*. PhD thesis, Palo Alto, CA: Stanford University.

Craw, S. and Sleeman, D. (1990). The flexibility of speculative refinement. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 28–32. Evanston, IL.

Dick, W. and Carey, L. (1990). *The systematic design of instruction*. Glenview, IL: Scott, Foresman/Little, Brown Higher Education. Third edition.

Gilmore, D. and Self, J. (1988). The application of machine learning to intelligent tutoring systems. In Self, J., editor, *Artificial Intelligence and Human Learning*, chapter 11. New York, NY: Chapman and Hall.

Ginsberg, A. (1990). Theory reduction, theory revision, and retranslation. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 777–782. Detroit, MI.

Johnson, W. L. (1986). *Intention-Based Diagnosis of Novice Programming Errors*. London: Pitman Publishing.

Langley, P., Ohlsson, S., and Sage, S. (1984). A machine learning approach to student modeling. Technical Report CMU-RI-TR-84-7, Pittsburgh, PA.: Carnegie-Mellon University.

Laubsch, J. H. (1975). Some thoughts about representing knowledge in instructional systems. In *Proceedings of the Fourth International Joint conference on Artificial intelligence*, pages 122–125.

Legree, P. J. and Gillis, P. D. (1991). Product effectiveness evaluation criteria for intelligent tutoring systems. *Journal of Computer-Based Instruction*, 18(2):57–62.

Michalski, R. S. (1983). A theory and methodology of inductive learning. In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors, *Machine Learning: An Artificial Intelligence Approach*, pages 83–134. Tioga.

Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, 18(2):203–226.

Murphy, M. A. and Davidson, G. V. (1991). Computer-based adaptive instruction: Effects of learner control on concept learning. *Journal of Computer-Based Instruction*, 18(2):51–56.

Nicolson, R. I. (1992). Diagnosis can help in intelligent tutoring. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, pages 635–640. Bloomington, IN.

Ohlsson, S. and Langley, P. (1985). Identifying solution paths in cognitive diagnosis. Technical Report CMU-RI-TR-85-2, Pittsburgh, PA.: Carnegie-Mellon University.

Ourston, D. (1991). *Using Explanation-Based and Empirical Methods in Theory Revision.* PhD thesis, Austin, TX: University of Texas. Also appears as Artificial Intelligence Laboratory Technical Report AI 91-164.

Ourston, D. and Mooney, R. (1990). Changing the rules: A comprehensive approach to theory refinement. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 815–820. Detroit, MI.

Ourston, D. and Mooney, R. J. (in press). Theory refinement combining analytical and empirical methods. *Artificial Intelligence*.

Plotkin, G. D. (1970). A note on inductive generalization. In Meltzer, B. and Michie, D., editors, *Machine Intelligence (Vol. 5)*. New York: Elsevier North-Holland.

Quinlan, J. (1990). Learning logical definitions from relations. *Machine Learning*, 5(3):239–266.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.

Reiser, B. J., Anderson, J. R., and Farrell, R. G. (1985). Dynamic student modeling in an intellignet tutor for LISP programming. In *Proceedings of the Ninth International Joint conference on Artificial intelligence*, pages 8–14. Los Angeles, CA.

Rich, E. (1990). Stereotypes and user modeling. In Frederiksen, N., Glaser, R., Lesgold, A., and Shafto, M., editors, *Diagnostic Monitoring of Skill and Knowledge Acquisition*, chapter 2. Lawrence Erlbaum Associates.

Rumelhart, D. E., Hinton, G. E., and Williams, J. R. (1986). Learning internal representations by error propagation. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing, Vol. I*, pages 318–362. Cambridge, MA: MIT Press.

Sleeman, D., Hirsh, H., Ellery, I., and Kim, I. (1990). Extending domain theories: two case studies in student modeling. *Machine Learning*, 5:11–37.

Sleeman, D., Kelly, A. E., Martinak, R., Ward, R. D., and Moore, J. (1987). Studies in the diagnosis and remediation of high school algebra students. *Cognitive Science*, 13:551–568.

Sleeman, D. H. and Smith, M. J. (1981). Modelling students' problem solving. *Artificial Intelligence*, 16:171–187.

Towell, G. and Shavlik, J. (1991). Refining symbolic knowledge using neural networks. In *Proceedings of the International Workshop on Multistrategy Learning*, pages 257–272. Harper's Ferry, W.Va.

Towell, G. G., Shavlik, J. W., and Noordewier, M. O. (1990). Refinement of approximate domain theories by knowledge-based artificial neural networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 861–866. Boston, MA.

VanLehn, K. (1983). *Felicity conditions for human skill acquisition: validating an AI-based theory*. PhD thesis, Cambridge, MA: Massachusetts Institute of Technology.

Wenger, E. (1987). *Artificial Intelligence and Tutoring Systems*. Los Altos, CA: Morgan Kaufmann.

Winston, P. H. and Horn, B. K. P. (1989). *Lisp*. Reading, MA: Addison-Wesley.

Young, R. M. and O'Shea, T. (1981). Errors in children's subtraction. *Cognitive Science*, 5:153–177.