Machine Learning, 5, 11-37 (1990)

# Extending Domain Theories: Two Case Studies in Student Modeling

D. SLEEMAN (SLEEMAN%CS.ABDN.AC.UK@NSFNET-RELAY.AC.UK)
*Computing Science Department, King's College, The University of Aberdeen, Aberdeen AB9 2UB, Scotland, UK.*

HAYM HIRSH* (HIRSH@SUMEX-AIM.STANFORD.EDU)
*Knowledge Systems Laboratory, Computer Science Department, Stanford University, Palo Alto, CA 94304*

IAN ELLERY (IE%CS.ABDN.AC.UK@NSFNET-RELAY.AC.UK)
*Computing Science Department, King's College, The University of Aberdeen, Aberdeen AB9 2UB, Scotland, UK.*

IN-YUNG KIM (KIM@UNIXB.CEL.FMC.COM)
*FMC AI Center, Central Engineering Laboratories, 1205 Coleman Avenue, Santa Clara, CA 95052*

Editor: Pat Langley

**Abstract.** By its very nature, artificial intelligence is concerned with investigating topics that are ill-defined and ill-understood. This paper describes two approaches to expanding a good but incomplete theory of a domain. The first uses the domain theory as far as possible and fills in specific gaps in the reasoning process, generalizing the suggested missing steps and adding them to the domain theory. The second takes existing operators of the domain theory and applies perturbations to form new plausible operators for the theory. The specific domain to which these techniques have been applied is high-school algebra problems. The domain theory is represented as operators corresponding to algebraic manipulations, and the problem of expanding the domain theory becomes one of discovering new algebraic operators. The general framework used is one of generate and test—generating new operators for the domain and using tests to filter out unreasonable ones. The paper compares two algorithms, INFER* and MALGEN, examining their performance on actual data collected in two Scottish schools and concluding with a critical discussion of the two methods.

**Keywords.** Incomplete domain theories, generate and test, failure-driven learning, explanation-based learning, student modeling, intelligent tutoring systems.

## 1. Introduction

Programming is frequently used by AI workers as a means of making a particular domain theory more explicit. The investigator programs that part of the domain that he believes he understands, and then contrasts the program's results on a predefined set of tasks with the expected performance. A thorough analysis of the discrepancies between the actual and anticipated performance (as well as the cases *not* covered) frequently helps the investigator see where the domain theory needs refinement. If such a theory is insufficient to solve the predefined tasks, it is said to be *incomplete*, and the theory needs to be extended. In the context of instruction (the application domain for this paper), a domain theory is incomplete

*Current Address: Computer Science Department, Rutgers University, Hill Center, Busch Campus, New Brunswick, NJ 08903 if it does not have a complete set of rules to cover all student errors. Thus a set of rules that are only sufficient to solve tasks *correctly* still constitutes an incomplete theory, since it will never cover an errant "solution."

We assume the use of the state-space paradigm for problem solving and theory representation. A domain theory is thus a set of operators, and if relevant operators are missing, the domain theory is incomplete and some tasks will not be solvable.[1] In this paper, we address the issue of having the *system* infer missing operators from the initial formulation of the domain theory. We describe two systems that differ in their response to this issue. One (INFER*) is used when operators are found to be missing during problem solving (when the system fails to solve a task). The other (MALGEN) produces a (more) complete set of operators *before* problem solving begins, expanding the initial domain theory before attempting to solve a task.

The domain in which these issues are explored is the modeling of student solutions of high-school algebra problems. An algebra equation is viewed as a state, and algebraic manipulations, perhaps including incorrect manipulations, are operators that transform one state (algebra problem) to another. The problem posed to the student is the initial state, and the final answer given by the student is the goal state. The problem of modeling a student's solution of an algebra problem thus becomes finding the sequence of operator applications that transform the posed problem, the initial state, to the student's answer, the goal state. Failure to model a student's solution demonstrates incompleteness in the domain theory, signaling the need for the generation of new operators.

When viewed in the state-space paradigm, problem solving is the process of finding a sequence of operator applications that transform an initial state into a goal state. The search for such operators can be viewed in a *generate and test* framework, as depicted in Figure 1. An *operator proposer* generates new operators; this generator may be constrained to limit the types of operators it produces. The candidate operators generated by the proposer are then tested by a *static filter,* which eliminates those that are not feasible. The set of new operators that pass through the filter, together with the existing operators, are used by the problem solver to solve future cases.[2]

This view of creating new operators does not constrain *a priori* the type of operators generated. For example, the new operators could be macro-operators, generated by composing existing operators. Heuristics used in this combinatorially explosive search include Iba's [1985] peak-to-peak heuristic. Often the number of macro-operators created is too large, and one needs some filter mechanism, such as that in Minton's [1985] MORRIS, to decide which should be retained. However, the present work focuses on the generation of missing operators that expand a domain theory: the INFER* and MALGEN systems. Because the

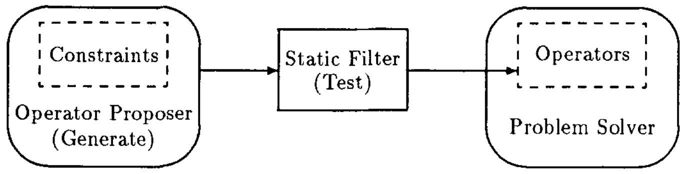

*Figure 1.* Framework for discovering new domain operators.

domain theory is viewed as operators in our framework, the task becomes one of implementing operator proposers and appropriate filters to remove some of the wilder proposals.

In the following section, we review some related work on extending domain theories, including earlier work in the area of student modeling. In Section 3, we describe PIXIE, the student modeling system that acts as the performance system in our studies. Sections 4 and 5 describe INFER* and MALGEN, two systems that extend PIXIE's domain theory for algebra to let it handle unexpected behavior. The following section evaluates the performance of the two approaches, and Section 7 summarizes the main points of the work, and discusses possible extensions.

## 2. Related work on extending domain theories

The use of domain expertise is central to many AI systems, and this has led to a variety of techniques for dealing with incomplete domain knowledge. For instance, *knowledge engineering* [Davis, 1979] involves querying the user for missing elements and interactively debugging the knowledge base by running a series of examples. In contrast, *nonmonotonic reasoning* works around missing information by using default rules [Reiter, 1980], circumscription [McCarthy, 1980], or default values [Minsky, 1975]. A third approach uses *analogy* [Carbonell, 1986] to reason from past solutions, letting one solve new problems to which domain knowledge cannot be directly applied.

However, this paper is concerned primarily with automated methods for acquiring domain expertise. One such approach uses inductive learning methods to generate domain rules from positive and negative instances, but most work in this area [for example Larson and Michalski, 1977; Quinlan, 1986] has started with little or no domain knowledge. Our approach differs in that it assumes one starts with a partial domain theory and extends this to account for new observations. Below we review two classes of systems that have taken this general approach.

### *2.1. Student modeling systems*

When an intelligent tutoring system attempts to correct the errant behavior of a student, it requires some representation of the student's current mastery of the domain. This is the problem addressed by student modeling systems, forming *models* of the student's ability based on the observed behavior of a student, which is usually in the form of student solutions to a set of known tasks. These models are built out of primitives provided to the system, and when they are insufficient to accurately model a student's behavior, the set of primitives must be extended. This is the task addressed by this paper.

Langley and Ohlsson's [1984] ACM is an example of a student modeling system. It starts with a set of operator-selection rules with overly general conditions. For each student the conditions are refined, so that the operators which are selected at any time are consistent with the student's behavior. This is done by exhaustively searching for a solution path that gives the same solution as the particular student. ACM then uses those operators lying on the path as positive instances of the various selection rules, and operators lying one step off the path as negative instances. The conditions on each rule are then refined using an ID3-like algorithm [Quinlan, 1986]. Hence ACM is altering existing rules rather than generating new ones.

VanLehn's [1987] SIERRA is a computational model that combines the essential aspects of both his Repair and Step theories. Repair theory seeks to give an explanation for students' ability to *complete* tasks, even when all the necessary procedures have not been learned; VanLehn postulates the existence of domain-independent repairs. Step theory attempts to explain how students *learn* incomplete rules when they are presented with correctly worked examples. Like Sleeman [1984], he argues that the student infers incorrect or incomplete rules from these worked examples. Further, he assumes that only *one* new subgoal is presented to the student at any time, thus simplifying the learning task.

Notice that ACM and SIERRA have very different objectives. The former produces a *diagnostic* model of an individual student's strategy, whereas the latter models the way in which a student learns.

## *2.2. Failure-driven learning systems*

If an AI system lacks essential knowledge, it will be unable to solve its task. At least two learning systems have addressed this problem by using details of the system's failure to determine the additional information necessary to solve the task. These systems thus use failure-driven learning to extend their theories.

Hall's [1988] PA system works in the domain of digital circuit design. His *precedent analysis* technique uses existing design rules to partially explain a given design, and proposes a new rule that would allow the completion of the design. Hall uses a hill-climbing approach for finding the smallest gap in completing a design, so that his technique need not address the problem of multiple possible completions. (As discussed later, INFER* instead finds all possible completions.) Hall also employs a *rule reanalysis* process to determine whether rules learned earlier can be simplified in terms of rules learned later (*cf.* Section 7).

Wilkins' [1987] ODYSSEUS learning apprentice attempts to explain actions taken by a domain expert using the knowledge of an existing expert system.[3] When it cannot create an explanation, the system forms new rules that will enable the underlying expert system to replicate the action when acting on its own. Rules must take on one of a fixed number of forms, and ODYSSEUS uses this knowledge to determine all explanations that could potentially exist. The explanation missing only one piece of knowledge is taken as the correct one, with an ordering on predicates determining which to select if there is more than one such explanation. ODYSSEUS uses a *confirmation theory* as its filtering process that determines whether the inferred rule is indeed correct and should be added to the knowledge of the expert system.

## 3. An overview of PIXIE

Before examining the INFER* and MALGEN systems in detail, we will describe in outline the PIXIE student modeling system[4] and the principal domain in which it has been used—high-school algebra. The goal of PIXIE is to model a student's problem-solving ability

*Table 1.* Typical student protocols: one correct (A) and two incorrect (B, C) protocols (student solution traces) for the task $3X + 5 = 23$.

| *A* | *B* | *C* |
|---|---|---|
| $3X + 5 = 23$ | $3X + 5 = 23$ | $3X + 5 = 23$ |
| $\downarrow$ | $\downarrow$ | $\downarrow$ |
| $3X = 23 - 5$ | $3X = 23 + 5$ | $8X = 23$ |
| $\downarrow$ | $\downarrow$ | $\downarrow$ |
| $3X = 18$ | $3X = 28$ | $X = {}^{23}/_{8}$ |
| $\downarrow$ | $\downarrow$ | $\downarrow$ |
| $X = {}^{18}/_{3}$ | $X = {}^{28}/_{3}$ | $X = 2{}^{7}/_{8}$ |
| $\downarrow$ | $\downarrow$ | |
| $X = 6$ | $X = 9{}^{1}/_{3}$ | |

and to provide appropriate remediation to improve the student's performance. Ideally, the system will respond to a student's input in well under a second. However, the model generation phase is computationally expensive, and thus potential student models are created during an *off-line* phase. When the student is interacting with the on-line system, the answer provided by the student is simply compared against a precomputed answer list.

PIXIE represents its domain operators as rules, some of which may be incorrect operators called *mal-rules*. The off-line sub-system generates a *model space* for each type of predefined problem—using correct rules, previously encountered mal-rules, and other information about the domain provided by the investigator. This set of rules can be viewed as PIXIE's domain theory, and through the remainder of this paper, we will use the terms *domain theory, domain rules,* and *rule set* interchangeably.

The main applications of PIXIE have focused on the domain of high-school algebra. For example, Table 1 shows the solutions to the problem $3X + 5 = 23$ for three different students. Protocol *A* represents a correct solution; *B* is a solution in which the student moved an integer to the other side of the equation without changing the sign; and *C* is a case in which the student introduced a major error of combining an X-term with an integer. PIXIE can only classify answers if the appropriate rules have been encoded for use in the model generation phase. If the system lacks the appropriate rules, it is unable to produce a model for the student.

In our earlier work we found mal-rules like those representing the errors above by carrying out detailed clinical interviews—a very labor-intensive process. The mal-rules encountered in this way include:

$$mX + n = p \rightarrow mX = p + n$$
$$mX + n = p \rightarrow qX = p$$
$$\quad \text{where } q = m + n$$
$$mX + n = p \rightarrow X + q = p$$
$$\quad \text{where } q = m + n$$
$$mX + nX = p \rightarrow X + X + q = p$$
$$\quad \text{where } q = m + n$$

where $m$, $n$, $p$, and $q$ are all integers, and where the first two mal-rules represent the errors noted in B and C in Table 1.[5] The objective of the systems described in the following sections is to partially automate the process of discovering these missing rules.

## 4. The INFER* system

The idea behind INFER* is that when a complete operator sequence from initial state to goal state cannot be found, one should propose a rule that would fill the *gap.* The system applies rules *forward* from the initial state, and *backward* from the student's answer, attempting to connect each node generated in the forward direction with the closest node(s) generated in the backward direction. It removes the assumption taken by its predecessor, INFER (Sleeman, 1982), that missing rules always occur as a first step in the student's solution path. As originally used, INFER applied reverse forms of rules to a student's answer, until either a form similar to the initial problem was reached or no further rules could be used, at which point its rule-inference step tried to form a rule to complete the missing last step. As INFER* uses a bi-directional search, it generates a larger number of nodes. For each pair of nodes generated in this space, it uses a rule-inference sub-algorithm to see whether a viable mal-rule between the two nodes can be generated.

This process can be stated more formally: *T-nodes* (target nodes) are generated by applying forward operators to the initial task, whereas *S-nodes* (source nodes) are generated by applying operators backwards from the student's answer. The S-nodes and T-nodes created for a given initial state and goal state along with the connections between them is referred to as the *S-T graph.* INFER* compares T-nodes with S-nodes, and calls the rule-inference sub-algorithm to look for all possible connections between heuristically selected S-T pairs. Figure 2 shows an abstract S-T graph.

### *4.1. An introductory example of the INFER* system*

For ease of experimentation, the INFER* system includes a set of system parameters. Specifically, it is possible to run the system either with or without the forward rule set. In the absence of this rule set, INFER* generates a series of nodes by working *backwards* from the student answer and then attempting to form new rules between those S-nodes and the *original* target equation. In this mode, the algorithm behaves exactly as the older INFER system (Sleeman, 1982). For the sake of clarity we will first discuss an initial run of the system using this degenerate mode.

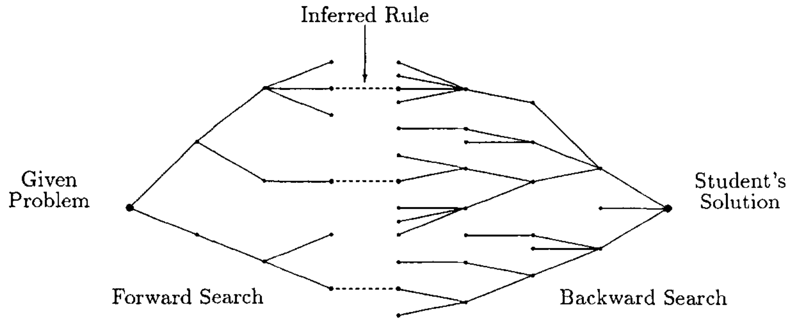

*Figure 2.* A schematic S-T graph.

Table 2. Using INFER* to learn a mal-rule given the task $3X + 5 = 6$ and the pupil's response $X = -2$.

| | | |
|---|---|---|
| a) | $X = -2$ | |
| | $\downarrow$ | BS1 [AddSub] |
| b) | $X = [-2 - n] + [n]$ | |
| | $\downarrow$ | Instantiate to RHS of target equation |
| c) | $X = 6 - 8$ | |
| | $\downarrow$ | B [NtoRHS] |
| d) | $X + 8 = 6$ | |
| | $\downarrow$ | Rule-inference step |
| e) | $3X + 5 = 6$ | |

Table 2 shows a successful path created by INFER* for inferring a mal-rule cited earlier, namely:

$$mX + n = p \rightarrow X + q = p,$$
$$\text{where } q = m + n.$$

As can be seen from the table, this process involves four basic steps:

1. Using domain rules in the backward direction, to transform an equation to something potentially closer to the initial problem. For example, B[NtoRHS], used between steps C and D in Table 2, is the use of NtoRHS in the *backward* direction, where NtoRHS is a rule that moves an integer from the left-hand side to the right-hand side of an equation and changes its sign.
2. Using *backward symbolic* rules that replace a number in a state by a *symbolic* expression. For example, between steps A and B, the $-2$ is replaced by $[-2 - n] + [n]$. In this case the responsible rule is BS1 [AddSub], where AddSub is a forward rule used to add or subtract two integers.
3. Using a series of heuristics to transform the symbolic expressions generated by backward symbolic rules into numerical expressions. The heuristics select values based on the numbers that actually occur on the left-hand and right-hand sides of the target equation. The rule applied between steps B and C is an example of such a heuristic.
4. Attempting to create a rule to complete the path when none of the above actions succeed. Such a rule is created between steps D and E in the table; we refer to this as the rule-inference step.

The current system incorporates 12 heuristics and backward-symbolic rules. Table 3 presents a selection of them.

These symbolic rules are necessary since there are an infinite number of integer pairs that sum to any given integer, and a method is needed for selecting the most suitable pairs. More generally, arithmetic operators can usually be applied in reverse in an infinite number of ways, but only a limited number of them are reasonable. Instead of the rule applying the reverse of an arithmetic operator to an integer $i$ to get two numbers $a$ and $b$, the reverse application results in a symbolic form that must be instantiated with specific numbers, which is left as a separate task. This latter task is performed by a heuristic rule, which is applied

*Table 3.* Simplified versions of a selection of backward-symbolic rules. "Target" is short for the target equation, such as $3X + 5 = 6$ in Table 2, and *eqn* is short for the *current* equation, which in Table 2 is *initially* $X = -2$.

```
BS1[AddSub]:
    IF lhs(eqn) $\neq$ lhs(target)
        AND rhs(eqn) $\neq$ rhs(target)
        AND rhs(eqn) = $i$
    THEN replace $(i,[i - ``n''] + [``n''])$

BS2[AddSub]:
    IF lhs(eqn) = $iX$
        AND contains(lhs(target), $jX + kX$)
    THEN replace $(i,([i - ``n''] + [``n'']))$

BS1[Mult]:
    IF lhs(eqn) $\neq$ lhs(target)
        AND rhs(eqn) $\neq$ rhs(target)
        AND rhs(eqn = $i$
    THEN replace$(i,[i \times ``n'']/[``n''])$
```

to select the appropriate instantiation for $n$ in a symbolic expression, such as $-2-n$ or $n$, based on information in the target state (Step 3 above). The heuristic rules *Instantiate to RHS of target equation* and *Instantiate to LHS of target equation* both select values for $n$ based on the numbers on one of the sides of the initial equation posed to the student. Thus instead of an infinite number of integer pairs that satisfy the relationship $a + b = i$, the set is usually reduced to only a few plausible pairs.

In addition, the conditions on BS1 [AddSub] and BS2[AddSub] result in selective application of the backward rules so they are not applied to every integer in the state. These rules should be contrasted with the *Instantiate to RHS of target equation* and *Instantiate to LHS of target equation* heuristics.

Notice also that Table 2 only shows a *successful* path for inferring a new rule. In general, *several* rules are satisfied by a particular state; for example, both BS1 [AddSub] and BS1 [Mult] are satisfied by A. Also, a rule can be instantiated in several ways; for example B[NtoRHS] can be used at C to move either the 6 or −8. Both factors can lead to considerable search,[6] which INFER* organizes using a breadth-first search algorithm.

In the rule-inference step in Table 2 we saw that INFER* learns the mal-rule

$$mX + n = p \rightarrow X + q = p,$$
$$\text{where } q = m + n.$$

However, as the algorithm notes which symbols have remained *unchanged*, it is possible to infer that the substantive change is

$$mX + n = string1 \rightarrow X + q = string1$$
$$\text{where } q = m + n.$$

Neves [1978] reports using a similar approach to deduce the *essential* features of rules given the several steps of a solution culled from an algebra text. For the moment we simply report the result of the rule-inference step; Section 4.3 gives some details of its execution.

*Table 4.* Part of the search tree working backwards from $X = 8$ to $5X + 3X = 24$.

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| $X = 8$ | $X = 8$ | $X = 8$ | $X = 8$ | $X = 8$ |
| $\downarrow$ | $\downarrow$ | $\downarrow$ | $\downarrow$ | $\downarrow$ |
| $5X = 40$ | $3X = 24$ | $2X = 16$ | $3X = 24$ | $X = 24 - 16$ |
| $\downarrow$ | | $\downarrow$ | | $\downarrow$ |
| $5X = 24 + 16$ | | $2X = 24 - 8$ | | $X + 16 = 24$ |
| $\downarrow$ | | $\downarrow$ | | |
| $5X - 16 = 24$ | | $2X + 8 = 24$ | | |
| | | $\downarrow$ | | |
| | | $X + X + 8 = 24$ | | |
| | | $\downarrow$ | | |
| | | $5X + 3X = 24$ | | |

## *4.2. Using INFER* without a forward rule set*

In general, a range of focusing heuristics are applicable and hence the algorithm usually generates multiple mal-rules. Table 4 shows part of the search required for inference of the mal-rule

$$mX + nX = p \rightarrow X + X + q = p,$$

or its more general form

$$mX + nX = \text{string1} \rightarrow X + X + q = \text{string1}, \quad \text{where } q = m + n,$$

given the initial task of $5X + 3X = 24$ and the student solution $X = 8$. The table shows several paths, only one of which (path 3) is successful. In this example, the focus is on the general nature of the heuristic used to transform the student's solution into the second node on the respective paths, and hence the level of detail is less than those of earlier traces, such as Table 2.

The first path starts by multiplying both sides of the equation by the coefficient of one of the X-terms in the target equation (in this case 5). This fails when a form similar to the target equation is reached and INFER* is unable to find a new rule for the final step. Paths 2 and 4 are identical; however, the first results from multiplying both sides of the equation by the other coefficient of the X-terms in the target equation (this time 3) and the second results from multiplying both sides of the equation by the factor

$$\frac{\text{right-hand side of target}}{\text{right-hand side of student equation}}.$$

Further progress halts on both paths for lack of an appropriate operator. Path 5 starts with the combined applications of BS1 [AddSub] with *Instantiate to RHS of target equation*, but fails to find any rules for its final state.[7] Finally, the third column shows the successful path that starts by multiplying both sides of the equation by the *number* of X-terms in the target equation.

## *4.3. Constraints on the rule inference step*

The rule-inference algorithm forms a rule only if a numerical relationship can be found between the coefficients of the target and current equations, that is, the T and S nodes. As there are an infinite number of numerical relationships between any set of integers, the initial search has been constrained such that:

1. each operand can occur at most once; and
2. the division of any two integers is rejected if the result is *not* an integer.

The first constraint can be relaxed so that operands may be used more than once, if no relationships are found under the constraints. There is no restriction on the number of times a particular operator may be used.

These constraints rule out a number of unlikely combinations. For instance, if one wished to find all combinations of 3 and 4 that make 7, then one might consider

$$3 + 4;\ 3 \times 4 - 4 - 4/4;\ 3/3 + 3/3 + 3/3 + 3/3 + 3/3 + 3/3 + 3/3;$$

and so forth. However, the constraints given earlier would exclude all but the first combination, that is all the *unreasonable* ones. There are situations, though, where one may need to relax these constraints considerably. For example, we have seen students solve tasks as

$$\begin{array}{ll} -3 \times 4 \rightarrow 9 & -3 \times 5 \rightarrow 12 \\ -4 \times 5 \rightarrow 16 & -2 \times 7 \rightarrow 12. \end{array}$$

From interviewing students, we know that they worked the task $-m \times n$ as $m \times n - m$. To enable INFER* to discover this we would have to allow $m$ to be used twice.[8] Deciding which constraints to use is a complex process that is based on considerable knowledge. We hope to capture this expertise; the constraints listed earlier are a first pass. Some more complex modes of the system are discussed in the next subsection.

## *4.4. Using INFER* with a forward rule set*

The original INFER algorithm [Sleeman, 1982] assumed that the inference step is always the first one in the student's solution path. As the student protocol in Table 5 illustrates, this does not always hold; in such cases the INFER algorithm would be unable to determine the unknown mal-rule. This was a major motivation for INFER*, which uses all known rules (both correct and incorrect) *forward* from the initial task, together with the backwards-chaining approach discussed above.

*Table 5.* Sample of a protocol with a mal-rule used at the fourth step of the solution ($mX = n \rightarrow X = n - m$).

| |
|---|
| $20X = 3[2X + 5]$ |
| $\downarrow$ |
| $20X = 6X + 15$ |
| $\downarrow$ |
| $20X - 6X = 15$ |
| $\downarrow$ |
| $14X = 15$ |
| $\downarrow$ |
| $X = 1$ |

Tables 6, 7, 8, and 9 show all the reconstructed protocols created by INFER* from the original task $20X = 3(2X + 5)$, given the student's answer $X = 1$. In each protocol FORWARD marks the states reached using forward operators from the initial task, and BACKWARD marks the states reached using backward operators from the student's solution. In all four protocols of Table 6, INFER* worked back from the student's answer $X = 1$ to $20X = -1 + 21X$; no rules were applied in the forward direction. The rule inference sub-algorithm then created four mal-rules between the new node and the original equation $20X = 3(2X + 5)$.

*Table 6.* Protocols for inferring potential mal-rules for a problem $20X - 3(2X + 5)$, given the solution $X = 1$. No forward rules were applied.

| | | | |
|---|---|---|---|
| a) | FORWARD: | $20X = 3(2X + 5)$ | |
| | | $20X = (3/(2 - 5)) + (3 \times (5 + 2))X$ | (Suggested inference step) |
| | BACKWARD: | $20X = -1 + 21X$ | |
| | | $20X - 21X = -1$ | |
| | | $-1X = -1$ | |
| | | $X = 1$ | |
| b) | FORWARD: | $20X = 3(2X + 5)$ | |
| | | $20X = (2/(3 - 5)) + (3 \times (5 + 2))X$ | (Suggested inference step) |
| | BACKWARD: | $20X = -1 + 21X$ | |
| | | $20X - 21X = -1$ | |
| | | $-1X = -1$ | |
| | | $X = 1$ | |
| c) | FORWARD: | $20X = 3(2X + 5)$ | |
| | | $20X = (2 - 3) + (3 \times (5 + 2))X$ | (Suggested inference step) |
| | BACKWARD: | $20X = -1 + 21X$ | |
| | | $20X - 21X = -1$ | |
| | | $-1X = -1$ | |
| | | $X = 1$ | |
| d) | FORWARD: | $20X = 3(2X + 5)$ | |
| | | $20X = (5 - (2 \times 3)) + (3 \times (5 + 2))X$ | (Suggested inference step) |
| | BACKWARD: | $20X = -1 + 21X$ | |
| | | $20X - 21X = -1$ | |
| | | $-1X = -1$ | |
| | | $X = 1$ | |

This set of inferred rules suggested how the student might have transformed $20X = 3(2X + 5)$ into $20X = -1 + 21X$. This can be written more generally as

$$mX = q(rX + s) \rightarrow mX = n + pX,$$
$$\text{where } p = q \times (s + r) \text{ and either } n = q/(r - s) \text{ or } n = r/(q - s) \text{ or}$$
$$n = r - q \text{ or } n = s - q \times r.$$

In Table 7, the original task is expanded *forward* to $20X - 6X = 15$ by means of two correct domain rules and the backward rules transform the student solution to $X + 14 = 15$. The system finds a single relationship between these two nodes, resulting in a single mal-rule. Tables 8 and 9 show the *forward* solution of the task being taken several steps further and to mal-rules subsequently being inferred that are essentially the same as that derived in Table 7. In this example, mal-rules cannot be formed for any other pair of S and T nodes because there are no other combinations of the available numbers that can be permuted to give 1.

*Table 7.* Protocol for inferring potential mal-rule for the problem $20X = 3(2X + 5)$, given the solution $X = 1$. Two forward rules were applied in this case.

| | | |
|---|---|---|
| | $20X = 3(2X + 5)$ | |
| | $20X = 6X + 15$ | |
| FORWARD: | $20X - 6X = 15$ | |
| | $X + 20 - 6 = 15$ | (Suggested inference step) |
| BACKWARD: | $X + 14 = 15$ | |
| | $X = 15 - 14$ | |
| | $X = 1$ | |

*Table 8.* Protocol for inferring potential mal-rule for the problem $20X = 3(2X + 5)$, given the solution $X = 1$. Three forward rules were applied here.

| | | |
|---|---|---|
| | $20X = 3(2X + 5)$ | |
| | $20X = 6X + 15$ | |
| | $20X - 6X = 15$ | |
| FORWARD: | $14X = 15$ | |
| | | (Suggested inference step) |
| BACKWARD: | $X + 14 = 15$ | |
| | $X = 15 - 14$ | |
| | $X = 1$ | |

*Table 9.* Protocol for inferring potential mal-rule for the problem $20X = 3(2X + 5)$, given the solution $X = 1$. In this run, four forward rules were applied.

| | | |
|---|---|---|
| | $20X = 3(2X + 5)$ | |
| | $20X = 6X + 15$ | |
| | $20X - 6X = 15$ | |
| | $14X = 15$ | |
| | $14X/14 = 15/14$ | |
| FORWARD: | $X = 15/14$ | |
| | | (Suggested inference step) |
| | $X = 15 - 14$ | |
| BACKWARD: | $X = 1$ | |

The technique of working in both directions and filling in gaps focuses the system's attention, and hence, in a sense, acts as a constrained generator. However, even this mechanism has the potential for generating many new rules for each problem. Thus the method of finding *reasonable* numerical relationships between states is used additionally to filter out some implausible rules.

In the current system, all the remaining mal-rules are presented to the investigator, who decides which are plausible and which are not. In the case of Table 6, the straightforward criterion of simplicity of the mal-rule would have been sufficient to rule out the mal-rules produced, but in general this may not be true. We view the formulation of a body of expertise (constraints on rules or meta-knowledge) to reject potential mal-rules as an important next step in this project; this topic is discussed in Section 7.

As noted elsewhere [Sleeman, 1987], individual student errors are unfortunately not always stable. However, mal-rules that occur several times with either a single student or within the population are given greater credence than ones that only occur once.

## 5. The MALGEN system

Earlier work in intelligent tutoring systems has demonstrated that incorrect problem-solving performance can be represented as variations on correct behavior. Examples are Carr and Goldstein's [1977] overlays, in which incorrect behavior is viewed as omitting parts of correct behavior, as well as Sleeman and Smith's [1981] mal-rules and Brown and Burton's [1978] procedural nets, in which correct performance is modified to represent incorrect behavior. This is the approach MALGEN takes, attempting to form new problem-solving operators that represent incorrect problem-solving performance by modifying existing operators.

The new operators are formed by performing simple perturbations on existing operators. The perturbations themselves can be viewed as operators working in a space of possible domain operators. New perturbation operators can be added and old ones can be removed in much the same way that domain operators can be defined. Perturbations can be applied to the newly generated operators, generating more operators, continuing the process as long as desired.

### *5.1. Rule representation*

To enable perturbations to be somewhat domain independent, MALGEN uses a robust representation for problem-solving operators. An operator is represented as a rule with four parts: a pattern, correctness conditions, actions, and a result. A state must match a rule's *pattern* if the rule is to be considered for use on the given state. If the pattern matches the state, the variables in the pattern are assigned values for use in the rest of the rule. If the pattern matches the equation, *correctness conditions* are checked to determine if this is the appropriate operator to apply. The *actions* carry out local computations that generate the new state, and the *result* specifies the state to be returned after the actions have been completed. The effort has been to make a robust representation that is capable of handling varying domains in the same, consistent manner.

*Table 10.* The rule ADD, an algebra operator that takes two numbers surrounding a "+" and correctly adds them.

```
ADD:
    pattern:
        the equation matches
            (?STRING1 ?NUM1 + ?NUM2 ?STRING2)
        where ?NUM1 and ?NUM2 are numbers
    correctness conditions:
        ?STRING1 does not end with ×, and
        ?STRING1 does not end with /, and
        ?STRING1 does not end with a −, and
        ?STRING2 does not start with ×, and
        ?STRING2 does not start with /, and
        ?STRING2 does not start with a variable
    actions:
        ?NUM3 ← ?NUM1 + ?NUM2
    result:
        (?STRING1 ?NUM3 ?STRING2)
```

To illustrate this representation, Table 10 presents the rule for ADD, an operator that correctly adds two numbers. The rule is only relevant if two numbers in the state—the algebra equation—can be added. Thus the pattern for ADD is that "?NUM1 + ?NUM2" appears in the equation, where "?NUM1" and "?NUM2" are pattern-match variables that must bind to numbers. ADD's correctness conditions check precedence relations to make sure the rule is being applied in the correct situation, and its actions compute the actual sum of the two numbers bound in the pattern. Finally, the result of ADD is the original equation with some new "?NUM3" replacing "?NUM1 + ?NUM2", where "?NUM3" was computed in the actions as the sum of "?NUM1" and "?NUM2".

As a further example, Table 11 presents the rule NtoRHS, which subtracts a number from both sides of an equation, moving a number across the "=" and switching its sign. The pattern for this rule checks if a "?SIGN ?NUM" occurs on the left side of the "="

*Table 11.* Rule NtoRHS, an operator that moves a number across an equals sign and switches its sign.

```
NtoRHS:
    pattern:
        the equation matches
            (?STRING1 ?SIGN ?NUM ?STRING2 = ?RHS)
        where ?SIGN is + or −, and ?NUM is a number
    correctness conditions:
        ?STRING1 has balanced parenthesis, and
        ?STRING1 does not end with ×, and
        ?STRING1 does not end with a /, and
        ?STRING2 does not begin with ×, and
        ?STRING2 does not begin with /, and
        ?STRING2 does not begin with a variable
    actions:
        ?NEWSIGN ← switch sign of ?SIGN
    result:
        (?STRING1 ?STRING2 = ?RHS ?NEWSIGN ?NUM)
```

in the equation, where "?SIGN" is a "+" or a "−", and "?NUM" is a number. In a similar way, the rule's correctness conditions make sure it is correct to move the number across the "=" sign. The variable "?SIGN" gets negated to "?NEWSIGN" in the actions, and the new equation—the original equation with "?NEWSIGN ?NUM" appended to the right-hand side and "?SIGN ?NUM" removed from the left-hand side of the equation—is returned. The condition of any rule can be a full Boolean expression, with any combination of conjunctions, disjunctions, and negations.

## *5.2. MALGEN's perturbations*

As described in the previous subsection, operators are rules with four parts: pattern, correctness conditions, actions, and result, with all but the first open to deviation by MALGEN. Incorrect versions of correct operators will have the same pattern as the original operator and differ in one or more of the other three parts. This section describes the perturbations used by MALGEN.

MALGEN keeps a list of rules pending perturbation, and separately perturbs the correctness conditions, the actions, and the result, each resulting in a new rule. The system checks each of these newly-generated rules against the existing rules, and only retains those that are distinct. Thus, if an incorrect operator can be generated in two different manners, only one copy is saved, and if a perturbation results in a correct rule, MALGEN does not retain the duplicate copy. New rules are placed at the end of the list of rules pending perturbation, and the system stops when no further rules are pending.

*5.2.1. Perturbing correctness conditions.* Incorrect operators that differ in correctness conditions cause the specified actions to occur at an incorrect time or place. For example, an incorrect application of ADD might be to change "$2 + 3 \times 4$" to "$5 \times 4$", ignoring precedence of arithmetic operators. Another example is "$-2 + 3$" to "$-5$", where the minus sign is handled incorrectly. In both these examples a valid action is applied to an incorrect state. The correct rule for ADD would check for these circumstances, and rules representing the incorrect versions above would be similar to the correct ADD, but with correctness conditions that deviate from the correct ones.

With this in mind, MALGEN first perturbs the correctness conditions for all rules by negating them. If the conditions contain a disjunction "$A_1 \vee A_2 \vee \ldots \vee A_n$", where the $A_i$ are arbitrary Boolean expressions, it would be negated to give "$\neg A_1 \wedge \neg A_2 \wedge \ldots \wedge \neg A_n$". However, given a conjunction "$A_1 \wedge A_2 \wedge \ldots \wedge A_n$", the negation mechanism generates $n$ new conditions, and hence $n$ new incorrect rules, namely "$\neg A_1 \wedge A_2 \wedge \ldots \wedge A_n$". "$A_1 \wedge \neg A_2 \wedge \ldots \wedge A_n$", . . . , through "$A_1 \wedge A_2 \wedge \ldots \wedge \neg A_n$", with each new rule representing a different way the original rule can fail. Further, the negation mechanism works recursively on each of the conjuncts, generating even more possibilities if there are nested conjunctions. Table 10 shows a simplified version of ADD, the rule to add two numbers, and Table 12 shows an incorrect version of this operator, generated by perturbing the correctness conditions.[9]

Note that the mechanism of negating conditions has the same effect as removing conjuncts. Both allow a perturbed rule to apply in the same situations as the original rule,

*Table 12.* The rule M1ADD, which adds two numbers that should not be added, due to precedence of operators. It is generated by perturbing the correctness conditions of the operator that adds numbers correctly.

```
M1ADD:
        pattern:
                the equation matches
                        (?STRING1 ?NUM1 + ?NUM2 ?STRING2)
                where ?NUM1 and ?NUM2 are numbers
        correctness conditions:
                ?STRING1 does not end with ×, and
                ?STRING1 does not end with /, and
                ?STRING1 does not end with a −, and
                ?STRING2 does start with ×, and
                ?STRING2 does not start with /, and
                ?STRING2 does not start with a variable
        actions:
                ?NUM3 ← ?NUM1 + ?NUM2
        result:
                (?STRING1 ?NUM3 ?STRING2)
```

and additionally in *new* situations in which the deleted conjunct would have caused the original rule to fail. However, by negating the conjunct rather than removing it, the new situations in which the rule would have failed are isolated and addressable independently. As a result, the new rules created by perturbing the correctness conditions of a single rule are mutually exclusive.

*5.2.2. Perturbing actions.* MALGEN incorporates three forms of action perturbations: changing actions, changing arguments, and removing actions. The first of these uses a list of primitive actions, each with a list of other primitive actions that can be used as replacements in creating new rules. The *replace* perturbation of Table 14 gives a simplified form of this perturbation operator. MALGEN then forms new rules by copying a rule and switching all primitives to similar ones, one at a time. Changing arguments, the second form of perturbation, simply takes existing actions and switches their arguments, as shown in the *switch arguments* perturbation in the table. The last perturbation, removing actions, takes primitives with one argument and replaces them with the IDENTITY primitive, one whose value is just the argument itself. This is shown in a simplified form as the *remove* perturbation in Table 14. For example, it would perturb the operator NtoRHS (Table 11) to generate a new operator, M1NtoRHS (Table 13), that moves a number across the "=" but neglects to switch its sign.

*5.2.3. Perturbing results.* Perturbation of results takes the general form of the returned answer and modifies it slightly. The motivation for this technique comes directly from the observed performance of high-school algebra students, who sometimes write the result incorrectly. For example, some students transform the algebra problem $4x = 2$ into $x = 4/2$. The final *switch* perturbation rule of Table 14 switches the two arguments of a binary operation, generating the domain rule just described.

Note that these perturbations differ from those involving actions, which create new operators that represent miscalculations; here the perturbations create new operators that

*Table 13.* The rule M1NtoRHS, which moves a number across the "=" sign like NtoRHS, but neglects to switch its sign. It is generated by switching the negate action to the identity action.

```
M1NtoRHS:
          pattern:
                    the equation matches
                              (?STRING1 ?SIGN ?NUM ?STRING2 = ?RHS)
                    where ?SIGN is + or −, and ?NUM is a number
          correctness conditions:
                    ?STRING1 has balanced parenthesis, and
                    ?STRING1 does not end with ×, and
                    ?STRING1 does not end with /, and
                    ?STRING2 does not begin with ×, and
                    ?STRING2 does not begin with /, and
                    ?STRING2 does not begin with a variable
          actions:
                    ?NEWSIGN ← ?SIGN
          result:
                    (?STRING1 ?STRING2 = ?RHS ?NEWSIGN ?NUM)
```

*Table 14.* Perturbations to replace an action with a similar action, to switch the arguments of an action, to remove an action, and to rewrite an equation with a binary operation incorrectly.

```
Replace:
          IF
                    ?ACTION is used in the actions, and
                    ?ACTION2 is similar to ?ACTION
          THEN
                    replace ?ACTION with ?ACTION2

Switch Arguments:
          IF
                    ?ACTION is used in the actions, and
                    ?ACTION has two arguments, and
                    ?ACTION is not commutative
          THEN
                    switch the arguments of ?ACTION

Remove:
          IF
                    ?ACTION is used in the actions, and
                    ?ACTION has only one argument
          THEN
                    replace ?ACTION with the identity action

Switch
          IF
                    the result matches
                              (?STRING1 ?OPERAND1 ?OP ?OPERAND2 ?STRING2)
                    and ?OP is binary
          THEN
                    rewrite the result as
                              (?STRING1 ?OPERAND2 ?OP ?OPERAND1 ?STRING2)
```

represent incorrect manipulations. Thus MALGEN could generate two operators, one that would compute $x = 2/4$ and give the result $x = 2$ by calculating division incorrectly, and another that would convert $4x = 2$ to $x = 4/2$ by doing the algebraic manipulations incorrectly. Both will result in the same final answer, yet the underlying errors are substantially different: one is an incorrect calculation, and the other is an incorrect manipulation.

### *5.3. Discussion of MALGEN*

Algebra is a domain in which domain operators can be partitioned into groups of operators that are *similar.* MALGEN formalizes this notion by using perturbation operators to specify the differences between similar operators and to form new domain operators from existing operators. The more general form of this approach is to apply *transformations* that form plausible new domain operators from existing ones. MALGEN *perturbs* operators, but one might codify other higher-level simularities as transformations and use them in the same manner. As such, the problem is related to analogy, in which similarities between some domain operators are codified in operators and then applied to new domain operators. It also resembles Lenat's [1983] use of mutation operators in AM and EURISKO to explore a space of concepts by modifying existing concepts.

MALGEN only uses "reasonable" perturbations, and so only reasonable operators are produced. This makes MALGEN a constrained generator of new operators, when viewed in terms of generate and test. Furthermore, when the system generates a new operator it queries the user about whether to proceed further with that operator. Thus the user serves as a filter for MALGEN. However, an automated filter would prove useful, especially if the perturbations used were less reasonable, such as randomly deleting elements in operators, since this would create a space too large for a user to filter. Generating many implausible operators could pay off if it produced one new missing operator that would not otherwise be discovered, and if the implausible operators generated could be filtered out automatically.

An early goal of this work, which has not been met, was to generate incorrect operators when needed. When the modeling system that uses the domain rules failed to find a solution path for a given problem, MALGEN was to suggest a new operator [Hirsh, 1985]. This requires a means of generating new operators ordered by some measure of their value to the modeling system. In one method that was considered, the strength of a perturbation operator was updated when one of its results proved useful in the modeling system; new operators would be created using perturbation operators of higher strength. Unfortunately, the domain of perturbations proved more difficult to formalize than algebra, and the credit-assignment problem was not as easy as it first appeared. This remains an area for future research.

## 6. Experimental assessment

INFER* and MALGEN have been used to automate the process of expanding incomplete operator sets for high-school algebra. Prior to this work, a researcher would analyze protocols that PIXIE was unable to model and suggest new operators that would allow a successful explanation of the protocol. Thus, evaluation of INFER* and MALGEN requires comparing their performance to that of the human they are meant to replace.

*Table 15.* A breakdown of the unmodeled errors.

| School | Tasks | Mal-rule | Copy | Arithmetic | Guess | Arith/Mal-rule | Total |
|---|---|---|---|---|---|---|---|
| LL | 58 | 14.5 | 4 | 20.5 | 19 | 2 | 60 |
| PP | 25 | 4 | 2 | 9.5 | 4.5 | 7 | 27 |

### *6.1 Data and evaluation procedure*

We collected data by using PIXIE with groups of 21 and 29 students from two different schools, which we will refer to as LL and PP. The students from school LL were divided into three subgroups: model-based remediation, reteaching, and control; those from school PP were divided into subgroups for model-based remediation and reteaching. Sleeman [1987] provides the details of such experiments. For the purposes of this paper we have analyzed in detail PIXIE's response to the model-based remediation groups. For school LL, this subgroup consisted of 8 students who were presented with 392 tasks; due to the reworking of 4 tasks, 396 solutions were generated. Of these, 278 (70.20%) were worked correctly, 29 (7.32%) were not attempted, and 89 (22.47%) tasks were worked incorrectly. Of those worked incorrectly, 31 (7.82%) were modeled by PIXIE's current collection of mal-rules and 58 tasks (14.64%) were unmodeled. In school PP, 15 students were in the model-based remediation group; they were given 591 tasks, of which 3 were reworked, giving a total of 594 solutions. Of these 536 (90.24%) were worked correctly, 4 (0.67%) were not attempted, and 54 tasks (9.09%) were worked incorrectly. Of these 54, PIXIE modeled 29 (4.88%) and failed to model 25 (4.21%).

Table 15 shows the breakdown of the unmodeled errors for both the data sets. One investigator (Sleeman) suggested that for both the LL and PP sets, 2 errors were made on 2 items, thus making the number of errors occurring in sets LL and PP 60 and 27, respectively. When the investigator was unable to decide whether the error was due to a mal-rule or an arithmetic error, 0.5 was assigned to each class, hence the decimal entries in Table 15. A detailed look at the data shows that the investigator suggested the error might be due to a mal-rule in 16 cases for the data from school LL and only 5 times for school PP.[10] The mal-rules and sign dropping (coded as Arith/Mal-rule) account for 30.0% (18/60) and 44.4% (12/27) of the total errors for groups LL and PP, respectively.

For INFER*, we simply need to determine whether, when given the student's answer and the original task, the system proposes a set of mal-rules that include the one proposed by the investigator. Since MALGEN does not take problems as input, its evaluation is based on whether it would generate from an existing rule the mal-rule proposed by the investigator. Detailed examples will be taken from the data for school PP, shown in Table 16. This table contains *all* the unmodeled errors for school PP which the investigator thought were due to a mal-rule or a sign being dropped. Each row specifies a task, the answer given by the student, and the analysis given by the investigator.

### *6.2. Experimental results for INFER**

Before giving the results of the experiments with INFER*, we would like to highlight features of the INFER* system that are relevant to experimentation. As noted earlier, one can run the system in a variety of modes, including:

*Table 16.* Mal-rules proposed by investigator for data from school PP.

| | Task | Student's Answer | Investigator's Mal-rule |
|---|---|---|---|
| P3 | $2X + 3 = 8$ | $X = 3/2$ | Arith: $8 - 3 \rightarrow 3$, *or* Mal-rule: $mX + n = p \rightarrow X = n/m$ |
| P4 | $7X = 2(5X + 6)$ | $X = 22/7$ | Mal-rule: $2(5X + 6) \rightarrow 10 + 12$ |
| P5 | $5X = 8(4X + 6)$ | $X = 48/27$ | Mal-rule: $5X - 32X \rightarrow 27X$ |
| P6 | $3X + 4 = 19$ | $X = 11/3$ | Arith: $19 - 4 = 11$, *or* Mal-rule: $3X + 4 = 19 \rightarrow 3X = 19 - 4 - 4$ |
| P7 | $5X = 2(3 - 2)$ | $X = -2/5$ | Arith: $6 - 4 = -2$ |
| P8 | $2X = 3(2 + 3)$ | $X = 9/4$ | Mal-rule: $3(2 + 3) \rightarrow 6X + 9$ *and* Arith: $2X - 6X = 4X$ |
| P9 | $7X = 2(5X + 6)$ | $X = 4$ | Arith: $7X - 10X = 3X$ |
| P10 | $6X = 10(9X + 5)$ | $X = 50/84$ | Arith: $6X - 90X \rightarrow 84X$ |
| P15 | $6X = 10(9X + 5)$ | $X = 42/25$ | Mal-rule: $6X - 90X \rightarrow 84X$ *and* $42X = 25 \rightarrow X = 42/25$ |
| P19 | $6X = 10(9X + 5)$ | $X = 50/84$ | Mal-rule: $(-aX = b \rightarrow X = b/a)$ |

- using only the backward rule set (*cf.* INFER), or using both forward and backward rule sets;
- using complete rule sets with all previously encountered mal-rules, partial rule sets, or only correct rules; and
- using operands only once or a prespecified number of times in an expression.

Experimentation was done using INFER* with three different rule sets: backward and correct only, backward and forward with correct rules only, and complete backward and forward (including all mal-rules discovered prior to the development of INFER*). In all cases operands were used only once.

Table 17 presents detailed results for the shorter dataset, from school PP; later we give a summary of analogous data for the larger dataset. Three statistics are given for each of the three execution modes: the size of the search space generated, the number of mal-rules proposed, and the number of *spurious* mal-rules created (those that were *not* substantially the same as the investigator's mal-rule). Note that decisions about the equivalence of two mal-rules is somewhat subjective. Further, the number of spurious rules also needs some explanation: for most tasks several mal-rules are generated from several different parts of the graph.

*Table 17.* Performance of INFER* on data for school PP.

| | Backward and Correct Rule Set | | | Both Directions: Correct Rule Set | | | Both Directions: Complete Rule Set | | |
|---|---|---|---|---|---|---|---|---|---|
| Task | Search Space | # Rules | # Spur. | Search Space | # Rules | # Spur. | Search Space | # Rules | # Spur. |
| P3 | 14 | 0 | 0 | 102 | 0 | 0 | 5613 | 11 | 11 |
| P4 | 13 | 1 | 0 | 121 | 3 | 0 | 23928 | 9 | 5 |
| P5 | 12 | 0 | 0 | 104 | 1 | 0 | 23842 | 5 | 3 |
| P6 | 14 | 0 | 0 | 134 | 0 | 0 | 3842 | 14 | 9 |
| P7 | 5 | 3 | 2 | 67 | 3 | 2 | 2067 | 7 | 6 |
| P8 | 4 | 0 | 0 | 68 | 0 | 0 | 1879 | 6 | 6 |
| P9 | 43 | 5 | 5 | 287 | 6 | 5 | 42449 | 57 | 55 |
| P10 | 12 | 0 | 0 | 92 | 1 | 0 | 22190 | 2 | 0 |
| P15 | 48 | 0 | 0 | 98 | 0 | 0 | 22324 | 1 | 0 |
| P19 | 12 | 0 | 0 | 92 | 1 | 0 | 22190 | 2 | 0 |

In reality, there are many fewer *distinct* mal-rules; one could argue that the count should focus on distinct *classes* of mal-rules. However, such a classification remains subjective; Sleeman and Ellery (1988) give a complete listing of mal-rules inferred for the PP dataset.

Looking at the results of these experiments, the INFER* algorithm with only the backward rule set generated two of the suggested twelve mal-rules, whereas the backward and forward correct rule set generated six, and the complete backward and forward rule set generated nine of the mal-rules. However, note the very considerable increase in the size of the search spaces and the corresponding increase in the number of spurious mal-rules produced.

The mal-rule for the first task (P3) in Table 16 is strange, as it involves using only some of the coefficients in the original equation; this error could be alternatively explained by an arithmetic slip. The other mal-rules that were missed were those in P8, where the investigator suggested there were two mal-rules. Had INFER* been able to cope with multiple missing rules, then both rules would have been found, as both mal-rules had been found when they occurred singly. Note that the investigator also suggested two mal-rules for task P15. With the complete rule set, one of the mal-rules inherent in P15 is already encoded, and so only one remains to be found. (The other modes fail to find either of these mal-rules.)

For the 10 tasks included in the tables, INFER* suggested 114 mal-rules when running with the complete backward and forward rule sets. In addition to the tasks given in the tables above, on 4 additional tasks the system proposed *acceptable* mal-rules where the investigator originally proposed none. For example, with the backward and forward complete rule set for tasks P14 and P17, INFER* proposed the protocols and mal-rules shown in Table 18.

In summary, INFER* has produced acceptable mal-rules for all but tasks P3 and P8. Had these new mal-rules been included in the rule base, PIXIE would have diagnosed a further 8 of the previously undiagnosed tasks. Thus of the 54 tasks that students of school PP worked incorrectly, 37 (as opposed to 29) or 6.23% would have been diagnosed and 17 (as opposed to 25) or 2.86% would have remained undiagnosed. This is a change of just 1.35% when compared to *all* tasks worked, but it represents a 32.00% reduction in the number of undiagnosed tasks, and moreover represents a 37/39 (94.87%) diagnosis of tasks worked incorrectly because of mal-rules; the investigator suggested that 15 out of the 54 errors were due to guessing, copying, or arithmetic errors.

*Table 18.* Mal-rules suggested by INFER* for tasks P14 and P17.

| Task: | P14 | p17 |
|---|---|---|
| Problem: | $4+6X = 22$ | $7X = 5X + 17$ |
| Solution: | $X = 6$ | $X = 2$ |
| Proposed Protocol: | $4 + 6X = 22$ | $7X = 5X + 17$ |
| | $10X = 22$ | $X + 7 = 5X + 17$ |
| | $X = {}^{11}/_{5}$ | $X = 5X + 10$ |
| | $X = 6$ | $X = {}^{10}/_{5}$ |
| | | $X = 2$ |
| Proposed Mal-rules: | $X = b/a \rightarrow X = b - a$ | $X = aX + b \rightarrow X = b/a$ |

*Table 19.* Number of mal-rules diagnosed by PIXIE with new rule sets from INFER*.

| | original rule set only | original plus inferred from LL | original plus inferred from PP | original plus both LL and PP |
|---|---|---|---|---|
| LL | $^{31}/_{48}$ = 64.58% | $^{45}/_{48}$ =93.75% | $^{33}/_{48}$ = 68.75% | $^{45}/_{48}$ = 93.75% |
| PP | $^{29}/_{39}$ = 74.35% | $^{34}/_{39}$ = 87.18% | $^{37}/_{39}$ = 94.87% | $^{37}/_{39}$ =94.87% |

For the more extensive dataset for school LL we will merely present a summary:

1. There were 16 mal-rules and 2 dropped signs noted by the investigator. For one task 2 mal-rules were deemed necessary, and in another a mal-rule and an arithmetic error was required.
2. Of the 18 possible mal-rules, INFER* with the backward correct rule set inferred 7 mal-rules, with the backward and forward correct rule set it formed 13 mal-rules, and with the complete backward and forward rule sets it formed 14 mal-rules.
3. One of the mal-rules not inferred involved using an operand twice (for $3X = 5 \rightarrow X = 8/5$); the other two tasks involved multiple errors, in one case two mal-rules, in the other a mal-rule and an arithmetic slip.
4. For 13 other tasks where the investigator had not suggested a mal-rule, INFER* proposed mal-rules that were accepted as plausible by the investigator, as well as suggesting plausible alternative mal-rules for many of the other tasks.

If all the mal-rules produced by INFER* for the LL school been included in PIXIE's rule-set when it was diagnosing the protocols from school PP, then a further 5 of the previously undiagnosed tasks would have been covered. This represents an increase of just 0.84% over all tasks worked, but it constitutes a 20% (5/25) reduction in the number of undiagnosed tasks. Moreover, since we estimated that only 39 of the tasks could be attributed to mal-rules (the others being copying errors, guesses, etc.), this represents a 87.18% (34/39) diagnosis for errors involving mal-rules. Table 19 summarizes the diagnostic performance of PIXIE with a variety of rule sets.

## *6.3. Experimental results for MALGEN*

Unlike INFER*, MALGEN can only form new rules that are similar to existing rules. However, it generates all but four of the mal-rules from Table 16, failing to create the mal-rules for P3, P4, P6, and the first of the two for P8. The reason for this is clear: the ones missed are sufficiently far from an existing rule that they are outside of MALGEN's current search space. For task P3 the mal-rule is not similar to any existing rule, and indeed drops one of the operands. Similarly, P4 necessitates using 2 instead of $2 \times 6$, again dropping an operand. In P6 and P8 additional terms are added, in one case by repeating an operand, and in the other by adding an $X$. Such dropping and adding of terms is not included in the perturbation set of MALGEN, and thus such mal-rules cannot be generated.

As noted earlier, the perturbations are merely a set of operators, and one could include others for doing such addition and deletion of terms. However, such operators create much

*Table 20.* Number of mal-rules diagnosed by PIXIE with the new rule set from MALGEN

| | original only | original plus rules generated by MALGEN PP |
|---|---|---|
| LL | $^{31}/_{48} = 64.58\%$ | $^{40}/_{48} = 83.33\%$ |
| PP | $^{29}/_{39} = 74.35\%$ | $^{35}/_{39} = 89.74\%$ |

larger search spaces and, as was the case for INFER*, result in a more complex and costly search. New perturbation operators would require the development of filters to rule out most of the unreasonable proposed rules.

The situation with school LL is similar to that of school PP. In eight of the tasks, the mal-rules proposed by the investigator could not be generated by MALGEN. In one case this was due to the fact that an operand is used twice, and in the remaining seven cases an operand, such as an $X$, is dropped. In all the other cases the necessary mal-rule is similar to an existing rule or mal-rule, and can thus be generated by MALGEN. Table 20 shows the improvement in diagnostic ability using the expanded rule set. Thus MALGEN, as reported above, finds a high proportion of the missing mal-rules but does not do as well as INFER* when it uses both forward and backward rules.

## 7. Conclusions

Further work on INFER* should focus on the processes of finding a gap for a given task and creating a rule to fill such a gap. A better assessment of *closeness* between S-nodes and T-nodes would help constrain generation of mal-rules. Another improvement would use other techniques besides pattern matching—such as analogy—to determine whether there is a connection between two nodes; this would let the system use additional domain-specific knowledge.

From Table 16 it is clear that a previously undiscovered mal-rule can occur several times in a data set, as in P9, P10, and P15. Thus INFER* would be more effective if, as soon as it created a new mal-rule, all outstanding protocols were checked to see if they too could be explained by the new mal-rule. The system would then only need to process those that were still unexplained.

INFER* currently uses only the initial task and the student's answer. In some cases, the system's performance could be greatly enhanced if it were also given intermediary steps in the student's protocol; such information would greatly reduce the size of the search space. For example, consider the protocol:

$$\begin{aligned} 35X &= 3 + 4(2X + 5) \\ 35X &= 3 + 8X + 10 \\ 35X &= 8X + 3 + 10 \\ 35X &= 8X + 13 \\ 35X - 8X &= 13 \\ 27X &= 13 \\ X &= {}^{13}/_{27} \end{aligned}$$

*Figure 3.* A *solution trace* from the original task, T, to the solution, S, where the use of two unknown rules, $U_1$ and $U_2$, leads to the formation of a composite rule.

It would be simple for INFER* to decide that the steps from $35X = 3 + 8X + 10$ to the student's answer were error-free, and merely to look for errors between the first two steps.

When INFER* determines a gap to which it should apply the rule-inference step, it generates only a single rule that covers the gap. This means that even when a combination of rules would better explain a protocol, the inference step will only generate a single *composite* rule. However, once discovered, a new rule can then be included in the forward and backward rule sets, making INFER* progressively better at analyzing complex protocols. Indeed, reanalyzing earlier protocols after later rule discoveries could let INFER* *decompose* composite rules, as in Hall [1988]. To clarify this point, Figure 3 shows a solution path between the task, T, and the solution, S. In this case there are two unknown rules in the path. Node names are given above the nodes, and rules that apply between nodes are given under the link. INFER* works in both directions, back from S to node 7, and forward from T to node 3, attempting to infer a new rule to cover the gap from node 3 to node 7 (which would include rules $R_4$ and $R_{10}$). However, if one of the unknown mal-rules, say $U_1$, is inferred from a later problem—that is, in another protocol in which it is the only unknown rule—it should be possible to return to this protocol and infer $U_2$, since knowing $U_1$ would let INFER* work forward from node T to node 6.

A further possibility for a static filter for INFER* is motivated by the development of the half-order theory in Meta-DENDRAL [Buchanan & Feigenbaum, 1978]. A researcher provided a set of constraints on which bonds in a molecule can and cannot break to Meta-DENDRAL, and they were used to limit the possible cleavage rules generated. A similar approach can be used more generally. We have formulated two components of such a half-order theory for this domain. First, from our extensive observations of students' algebra, we believe students do have a strong idea of the acceptable form of an algebra answer. Thus, although we have seen many types of errors, we have *never* seen tasks of the form $mX = n$ changed to either $X = n/$ or $X = /m$. This suggests one possible type of filter, formalizing the notion of well-formed equations. One difficulty in this approach is that rules such as those implied above apply to the description of states rather than operators. Before such a filter could be used, it is necessary to determine which operators have the potential for creating a state that would fail the given condition. The second component suggested by the two pairs of possible algebra rules proposed in section 4.4 is that of simplicity. In both instances the new rule supported by (clinical) interviews was the simpler of the two hypotheses. However, a consistent definition of simplicity can be difficult to determine, and it may not always be appropriate. Thus, this approach to formalizing a static filter requires inspection of new rules proposed by the generator, and needs criticism of the rules by experts to suggest *why* certain rules are *not* acceptable. This knowledge then needs to be captured and represented as constraints.

Although the techniques used in INFER* are quite general, certain portions of the system could be made even more task independent. All the domain-specific information has been included in the knowledge base for the algebra domain. However, the algorithm that currently seeks to establish a (semantic) relationship between elements of the T- and S-nodes looks merely for a *numerical* relationship between the entities (that is, the coefficients). This function is probably domain specific, and in general INFER* will need to establish different types of relationships, such as the relationship between toenail and arm or the relationship between chlorine and halogen (Chisholm & Sleeman, 1979).

Finally, an improvement in the efficiency of these systems can be achieved by segmenting the rule set and only working with the relevant subset at any time.

This paper has demonstrated the use of several techniques to extend a domain's operator set or theory. INFER* proposes *new* operators to bridge gaps during the reasoning process. On the other hand, MALGEN proposes new operators by *modifying* existing operators, using a set of perturbation rules. Although applications to date have focussed on algebra, we believe these techniques are applicable in other domains. With other knowledge bases, one might use MALGEN to generate an initial set of mal-rules and use INFER* to create the more rarified, idiosyncratic ones not suggested by the former, as MALGEN is computationally less demanding than INFER*. We predict that these techniques will become increasingly more important as knowledge bases in intelligent systems become larger and—virtually by definition—more inconsistent and incomplete.

## Acknowledgments

This work has been supported by a grant from IBM and indirectly by ARI/ONR Contract No. MDA-903-84-k-0279. Tektronix donated workstations to Stanford on which the INFER* project was originally implemented; other computing resources were provided by Stanford's SUMEX facility, funded in part under NIH Grant RR-00785 from the Division of Research Resources Biomedical Research Technology Program.

The pupils of Abbey Grange School, Leeds, England, provided some fascinating protocols. We thank Jaime Carbonell, Pat Langley, Bruce Buchanan, and the members of the GRAIL group at Stanford for discussions concerning various aspects of this work. Devika Subramanian, Paul Rosenbloom, David Wilkins, Bruce Buchanan, Pat Langley and the two reviewers provided valuable comments on earlier drafts of this paper. David Ericson implemented the original version of the algorithm that finds relationships between sets of integers. The data used to evaluate INFER* and MALGEN was collected in Aberdeen with assistance from R. Martinak, A.E. Kelly, Robert Ward, and Joyce Moore. We also thank the schools involved for their collaboration.

## Notes

1. This is not to say that the only form of incompleteness is missing operators. For example, a state description language that lacks relevant attributes can also cause incomplete performance.
2. This framework is only part of a more general scheme that includes a *dynamic filter*, to filter possibilities during problem solving, and filter compilation [Bennett & Dietterich, 1986]. Iba [1989] uses a similar framework in his work on macro-operator learning, and some of the terminology used here is taken from his paper.

3. ODYSSEUS has also been used as a student modeling mechanism that attempts to explain the behavior of a student when compared to an existing expert system. Both applications of ODYSSEUS have been explored in the domain of medicine.
4. See Sleeman [1983] for a detailed discussion of LMS, the precursor to PIXIE, and the underlying algorithm.
5. For a more complete list of mal-rules for the domain of high-school algebra, see Sleeman [1985].
6. Some of this branching could be avoided if the backward form of NtoRHS were made more *sophisticated* by allowing it to access the target equation; that is, using a heuristic form that is applied selectively, c.f., BS1[AddSub].
7. INFER* will not create a state that is more complicated (that is, involves more terms or symbols) than the target equation.
8. However, allowing 3 and 4 to be used twice to obtain 7 increases the number of possible combinations obtained from one to ten.
9. The *M* preceding operator names is taken form Sleeman's [1983] method of prefixing incorrect *mal-rules* with the letter *M*.
10. For the LL data, some 13 were unambiguous and 3 were ambiguous, hence a score in Table 15 of 14.5 (13 + 3 × 0.5), corresponding to 16 potential mal-rules. For the PP data, 3 were unambiguous and 2 were ambiguous, hence a score in Table 15 of 4 (3 + 2 × 0.5), which corresponds to 5 potential mal-rules.

## References

Bennett, J.S, & Dietterich, T.G. 1986. *The test incorporation hypothesis and the weak methods* (Technical Memo TR-86-30-4). Corvallis: Oregon State University, Computer Science Department.

Brown, J.S., & Burton, R. (1978). Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive Science, 2,* 155–198.

Buchanan, B.G., & Feigenbaum, E.A. (1978). DENDRAL and Meta-DENDRAL: Their application dimensions. *Artificial Intelligence, 11,* 5–24.

Carbonell, J.G. (1986). Analogy in problem solving. In R.S. Michalski, J.G. Carbonell, & T.M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). San Mateo, CA: Morgan Kaufmann.

Carr, B., & Goldstein, I. (1977). *Overlays: A theory of modeling for computer aided instruction* (AI Memo 406). Cambridge, MA: Massachusetts Institute of Technology, Laboratory for Artificial Intelligence.

Chisholm, I.H., & Sleeman, D.H. (1979). An aide for theory formation. In D. Michie (Ed.), *Expert systems in the microelectronic age.* Edinburgh: EUP.

Davis, R. (1979). Applications of meta-level knowledge to the construction, maintenance and use of large knowledge bases. In R. Davis & D. Lenat (Eds.), *Knowledge-based systems in artificial intelligence.* New York: McGraw-Hill.

Hall, R. (1988). Learning by failing to explain: Using partial explanations to learn in incomplete or intractable domains. *Machine Learning, 3,* 45–77.

Hirsh, H. (1985). *Modeling problem-solving performance.* Master's Thesis, Computer Science Department, Stanford University, Palo Alto, CA.

Iba, G.A. (1985). Learning by discovering macros in puzzle solving. *Proceedings of the Ninth International Joint Conference in Artificial Intelligence* (pp. 640–642). Los Angeles, CA: Morgan Kaufmann.

Iba, G.A. (1989). A heuristic approach to the discovery of macro-operators. *Machine Learning, 3,* 285–317.

Langley, P., & Ohlsson, S. (1984). Automated cognitive modeling. *Proceedings of the Fourth National Conference on Artificial Intelligence* (pp. 193–197). Austin, TX: Morgan Kaufmann.

Larson, J.B., & Michalski, R.S. (1977). Inductive inference of VL decision rules. *SIGART Newsletter, 63,* 38–44.

Lenat, D.B. (1983). The role of heuristics in learning by discovery: Three case studies. In R.S. Michalski, J.G. Carbonell, & T.M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* San Mateo, CA: Morgan Kaufmann.

McCarthy, J. (1980). Circumscription—A form of non-monotonic reasoning. *Artificial Intelligence, 13,* 27–39.

Minsky, M. (1975). A framework for representing knowledge. In P.H. Winston (Ed.), *The psychology of computer vision.* New York: McGraw-Hill.

Minton, S. (1985). Selectively generalizing plans for problem-solving. *Proceedings of the Ninth International Joint Conference in Artificial Intelligence* (pp. 596–599). Los Angeles, CA: Morgan Kaufmann.

Neves, D.M. (1978). A computer program that learns algebraic procedures by examining examples and working problems in a textbook. *Proceedings of the 2nd National Conference of the Canadian Society for Computational Studies of Intelligence.* Toronto, Canada.

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

Reiter, R. (1980). A logic for default reasoning. *Artificial Intelligence, 13,* 81–132.

Sleeman, D.H. (1982). Inferring (mal) rules from pupil's protocols. *Proceedings of the 1982 European AI Conference* (pp. 160–164). Orsay, France.

Sleeman, D.H. (1983). A rule directed modeling system. In R.S. Michalski, J.G. Carbonell, & T.M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* San Mateo, CA: Morgan Kaufmann.

Sleeman, D.H. (1984). An attempt to understand students' understanding of basic algebra. *Cognitive Science, 8,* 387–412.

Sleeman, D.H. (1985). Basic algebra revisited: A study with 14 year olds. *International Journal of Man-Machine Studies, 22,* 127–149.

Sleeman, D.H. (1987). Some challenges for intelligent tutoring systems. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (pp. 1166–1168). Milan, Italy: Morgan Kaufmann.

Sleeman, D.H., & Ellery, I. (1988). *A catalogue of mal-rules generated by INFER** (Technical Report). Aberdeen, Scotland: University of Aberdeen, Computing Science Department.

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

VanLehn, K. (1987) Learning one subprocedure per lesson. *Artificial Intelligence, 31,* 1–40.

Wilkins, D.C. (1987). *Apprenticeship learning techniques for knowledge-based systems.* Doctoral dissertation, University of Michigan, Ann Arbor, MI.