A Practical Authoring Shell for Apprenticeship Learning

William J. Clancey
Kurt Joerger
Institute for Research on Learning
Cimflex Teknowledge, Inc.

In *Toward a Scientific Practice of Science Education,* Gardner, et al., eds, Hillsdale: Lawrence Erlbaum, 1988, pp. 141-161. Revised from Proceedings of ITS-88, Montreal, June 1988.

Abstract

A novel, practical authoring and tutoring program is described that enables a teacher to adapt an existing expert system for use in teaching. The expert system is run in a normal manner by a student, typically using predetermined cases selected by the teacher. The expert system stops at breakpoints set during the authoring phase, and invokes a general tutoring program that probes the student's understanding of relevant data and intermediate conclusions. The authoring program analyzes the knowledge base and helps the teacher determine which breakpoints will produce meaningful interactions. Breakpoints can be varied for different cases and a library of cases can be built into a sequenced lesson curriculum. The authoring and tutoring systems are both rule-based programs, written in the same language as the knowledge base, making them easy to change and easy to interface with the expert system.

1. Introduction

One of the greatest instructional advances in the past decade has been the development of knowledge-based models of expert reasoning. For the first time, we are able to capture and replicate part of the process by which an experienced problem solver gathers information about a problem, forms intermediate abstractions summarizing the problem situation (e.g., diagnoses, design specifications) and relates this understanding to possible courses of action (e.g., repair plans, process control plans). In *knowledge-based tutoring*, a general instructional program can be used to interact with students using different knowledge bases (Clancey, 1987). Separating the domain knowledge base from the teaching knowledge provides tremendous engineering efficiency, allowing the teaching knowledge to be more clearly formalized, evaluated, and reused.

However, the knowledge-based tutoring paradigm has several practical difficulties. Among the most important is the requirement that the knowledge base be well-formed in order to interface properly with the general instructional program. Although knowledge acquisition programs might one day semi-automate this process, the reality today is that the endeavor is one that only experienced programmers (or experts trained to be programmers) can accomplish after many months of tedious design and debugging. Indeed, research of the past decade has only further increased our standards of knowledge representation desireable for teaching, while tools for constructing such programs lag far behind or are not generally available.

This leaves us with an immense practical problem for meeting the potential of knowledge-based tutoring. Are teachers simply to wait until knowledge acquisition programs become smarter? Can anything be done to provide useful capability on commercially available and supported tools, encouraging a more rapid application of our research, particularly in the universities? Is there a practical method for exploiting AI research in a simple way, perhaps

compromising some of the current research concerns focusing on knowledge representation and explanation capability, but nevertheless providing something useful for teachers in classrooms today?

With these questions in mind, we sought to develop a novel means for using expert systems for teaching, gaining some of the benefits right away for teachers with minimal resources or knowledge engineering capability. Based on other reported research (Bahill, 1986), that demonstrated that teachers were ready to develop programs for their classes, we decided to develop a shell for simple, rule-based expert systems. In particular, we sought to avoid the problems encountered in GUIDON, which required such a rule base to be well-formed, and put aside the ideals of NEOMYCIN (Clancey, 1986), which imposed epistemological distinctions that go beyond most teachers' experience and programming capabilities. We wanted an authoring program that could cope with messy knowledge bases--containing arbitrary program code in places--yet still carry on a meaningful interaction with a student.

Furthermore, based on requests from industrial users of expert systems, we sought to provide a mode for running a expert system that would enable to user to learn on the job, in the course of his routine operation of the program. The goal is to convey to the user some of the routine problem-solving knowledge in the program so he can use the program responsibly as a tool--better understanding how it operates--and build on the program's methods through his own experience. This is one form of *apprenticeship learning*, an instructional approach that emphasizes the importance of learning in the context of actual problems (Collins, et al., 1986). As will be seen, the environment we have designed provides other features of apprenticeship learning, including modeling the process of expert reasoning, helping the student through difficult areas (scaffolding), and sequencing problems and support over time (fading) (Collins, et al., 1986).

The key idea of our approach is to have a running expert system invoke the tutoring system at breakpoints set by a human teacher, in a way that can be customized for individual cases and students. That is, we transfer some of the burden of adapting the expert system to the teacher, who selects and orders the concepts and example cases that will be discussed with the students. On the other hand, the authoring program handles the technical details of determining which rules are compatible with the teaching program and preparing the necessary code that enables an interruption to occur during the expert system's operation. Furthermore, the teaching program handles all details concerning the form of probing appropriate to the situation, adapting to the role of the chosen concept in the current case and the student's response to its questions. Questioning itself follows a simple Socratic format, involving probing in more detail for incorrect responses, as applied in SCHOLAR and GUIDON.

This design is implemented in a working program called *Training Express* (tm).[1] The program is implemented in the M.1 (tm) knowledge base language and provides a teaching interaction with domain knowledge encoded in arbitrary M.1 knowledge bases, subject to some limits discussed below. The program has been integrated with a database system, dbase III+, which the authoring program uses to store a "curriculum" or library of cases to be run by a student, as well as records of student performance. An important part of the design is that the teacher can vary breakpoints from case-to-case, allowing probing to vary with problem difficulty. Furthermore, cases need only be partially specified, so a student can vary the input, exploring how the expert system behaves in different situations, while still being probed in a manner the teacher found compatible with the prespecified, fixed inputs.

Sections that follow provide scenarios of teacher and student interaction, a discussion of the novelties, advantages, and limitations of the approach, a discussion of the status and future development of the product, and conclusions about the practical development of AI research.

2. Scenarios: Teacher and Student Interactions

In this section we describe the basic operation of Training Express. For simplicity, we will continue to refer to the "authoring program" and the "teaching program," which are in fact completely separate pieces of code, implemented as two M.1 knowledge bases (see Figure 1). In this section, we first establish some basic terminology, describe the flow of control, and provide working examples of the program.

2.1. Basic operation

When a student using Training Express runs a new or canned case, the expert system invokes the teaching program at what we call *breakpoints*. Each breakpoint corresponds to a concept in the knowledge base (e.g., "the name of the disease requiring therapy," "the shape of the structure being analyzed"). Breakpoints are set by the human teacher in an interactive program called the *authoring system*.[2] This program reads the domain knowledge base and prepares a *secondary knowledge base*, which contains cross-index information--declarations written in the M.1 language that cause breakpoints to occur. When the teaching program is invoked, it asks

---

[1] Trademark of Teknowledge, Inc. Patents pending. The program runs on personal computers with a color monitor in 640K of resident memory. Interfaces are coded in "C."

[2] M.1 uses an expression-value language. Therefore an expression bundles an object (e.g., "structure being analyzed") with an attribute (e.g., "the shape"). This simplifies the knowledge representation task.

the student questions, which we call *probes*: What conclusions can be made now? What factors support these conclusions? What value for a given factor is consistent with the conclusions? The teaching program indicates which student answers are correct, wrong, or missing, probing in more detail when mistakes are made. Domain rules are referenced by name, so the student can examine the knowledge base, using already available M.1 commands like "list" and new commands supplied with the tutor, such as "evidence."

In summary, with more detail, the two M.1 programs constituting Training Express do the following:

- The authoring system:
  - interacts with a human teacher to determine which concepts (expressions) should be discussed with the student (breakpoints)
  - analyzes rules concluding these expressions
  - creates a secondary knowledge base with breakpoints marked
  - enables the teacher to edit and store text that introduces individual cases, explains the significance of individual breakpoints, provides a synonym for referring to the knowledge base concept, and defines the meaning of the knowledge base concept.
  - enables the teacher to store a library of cases, to be sequenced in a particular way, called a *curriculum*, for presentation to a student.
  - enables the teacher to examine and maintain a database of student records pertaining to their performance and knowledge exhibited when using the teaching program.
- The teaching program:
  - loads the domain knowledge base and secondary knowledge base created through interaction with the teacher
  - interacts with the student to select an appropriate case
  - interacts with the student during the expert system consultation to probe his understanding and provide definitions and explanations upon request.
  - stores a record of the student's performance and knowledge.

It is important to realize that an M.1 domain knowledge base can be used without changes. Furthermore, in the authoring system, the teacher simply selects breakpoints from a single multiple choice question. Everything else is handled automatically. Besides the secondary knowledge base, which provides cross-index information making the teaching program more efficient, the interface between the domain knowledge base and tutor consists of a single M.1 statement for each breakpoint, e.g.,

```
whenfound(wine = X) = whenfoundtutor(wine).
```

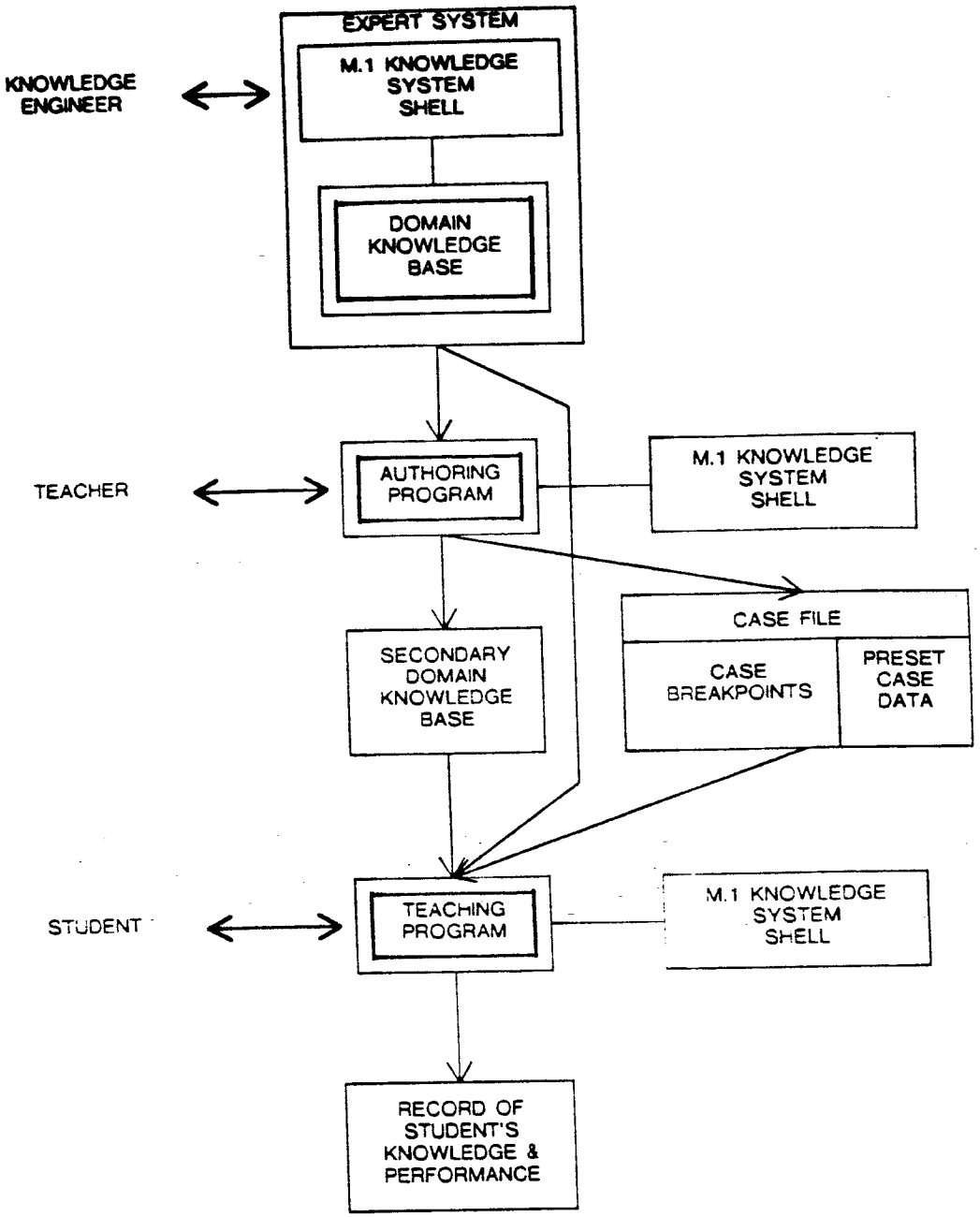

Figure 2-1: Basic flow of control: A knowledge engineer or teacher interacting with the M.1 shell creates a domain knowledge base; this knowledge base is loaded and analyzed by the authoring system, which interacts with the human teacher to set breakpoints, define cases, and provide additional text explanations; these files are then available to the teaching system, which interacts with a student while the expert system is in the normal course of solving a problem.

This specifies that when the M.1 expression "wine" has been determined by the expert system in the context of a particular case, with any value (indicated by the variable, X, but the conclusion cannot be "unknown"), the M.1 expression "whenfoundtutor(wine)" should be sought as the next goal. Thus an interruption occurs during the normal processing of the expert system. Rules concluding "whenfoundtutor(ANY)" match the new goal (here "whenfoundtutor(wine)"). These rules interact with the student, probing him about the expression "wine." Thus there are really two M.1 knowledge bases running together, the domain expert system and the teaching program. Before going into more detail, we present an example.

2.2. Authoring System Example Dialogue

The following examples demonstrate Training Express using a simplified version of SACON (Bennett, 1979), an expert system developed to aid civil engineers in setting up a complex software package that performs structural analysis (stress and deflection analysis of structure behaviors under particular loads). The task of SACON is to abstract the structure being analyzed, use heuristic rules to approximate the structure's behavior, and select one or more computer programs that will provide more detailed analysis of the expected behaviors.

```
M.1> author sacon
Loading the sacon knowledge base...done.

Analyzing the knowledge base for
expressions concluded by rules...done.

Which expressions would you like the
tutor to discuss with the student?
    1.  alpha←gamma(SS)
    2.  analysis←class(STRUCTURE)
    3.  analysis←recs(STRUCTURE)
    4.  cannot-provide-advice(STRUCTURE)
    5.  conclusions←printed
    6.  consultation←over
    7.  deflection(STRUCTURE)
    8.  deflection←bound(LOADING)
    9.  deflection←stress←magnitudes(LOADCOMP)
    10.  density(SS)
    11.  displayac(STRUCTURE)
    12.  displayar(STRUCTURE)
    13.  einertia(SS)
    14.  errorclass
    15.  ewidth(SS)
    16.  idata
    17.  length([X|L])
    18.  length←width←ratio(SS)
    19.  maximum(A,B)
    20.  nd←deflection(SS)
    21.  nd←stress(SS)
    22.  nonlinearity(STRUCTURE)
    23.  rms(NUMLIST)
```

```
    24.  shape(SUBSTRUCTURE)
    25.  sm←and←dm←parms(LC)
    26.  ss←deflection(SS)
    27.  ssnonlinearity(SS)
    28.  ss←stress(SS)
    29.  stress(STRUCTURE)
    30.  stress←bound(LOADING)
    31.  stress←criterion(SUBSTRUCTURE)
    32.  sum([FIRST|REST])
    33.  sumsquares([FIRST|REST])
    34.  youngs←modulus(SS)
>> 2,24,26,27,28
Reading the rules for analysis←class(STRUCTURE)...
rule-10...
rule-11...
rule-12...Error: Ignoring listof(stress(STRUCTURE))
rule-13...
rule-64...
rule-27...
done.

Reading the rules for shape(SUBSTRUCTURE)...
rule-54...
rule-55...
rule-56...
done.

Reading the rules for ss←deflection(SS)...
rule-58...
rule-59...
rule-60...
rule-50...Error: Ignoring listof(DB,loading(SS) = L and
                 deflection←bound(L) = DB)
rule-53...Error: Ignoring SQ/L
          Error: Unable to find values for variable
                 on right hand side of rule-53

        << section omitted >>

Save secondary knowledge base?
>> y
Saving the secondary knowledge base...done.

To run the tutor in a fresh M.1, load TREX.KB,
and give the command, "teach sacon."
```

Notice that the authoring program indicates which domain rules cannot be successfully analyzed and why not. With this information, the teacher can decide to omit the breakpoint or, if it is something important to teaching, rewrite or seek help for rewriting the offending rules. In practice, the offending rules or rule clauses concern a computation which is an artifact of the implementation; it can often be hidden by defining an intermediate expression whose value summarizes the computation. Indeed, we find an obvious correspondence: the clean, easily analyzed parts of the knowledge base tend to contain the basic associations we wish to convey to a student. The procedural and computational parts can be ignored by the teaching program. This strategy works especially well, given the approach of interrupting the

working expert system. That is, breakpoints can be deliberately defined to bypass difficult to explain computations and to focus on the essential domain facts and associations.

The specific limitations of the authoring system change over time as we add new capabilities to the analysis rules and as M.1 itself becomes more complex with new constructs that make analysis more difficult. The program has fairly extensive capability to deal with numeric computations involving variables, as occur in SACON. In the current version, the chief restrictions involve list manipulation. For example, rules concluding a value that is a list are flagged by the analysis program. A break will still occur, but the teaching program states conclusions, rather than probing.

Note that the authoring program is menu-driven with many other capabilities, as summarized above. Each case is conceived to be a *lesson*, in the sense that it brings out a certain set of concepts in a particular problem context. The teacher defines a case by running the domain expert system within Training Express, and using the menus to define problem data (called *presets*), define terms, provide text to appear when a breakpoint occurs, explanatory text for specific rules, etc. See Figure 2. Other menus are used to sequence lessons and provide introductory and concluding text. See Figure 3.

2.3. Teaching Program Example Dialogue

Here again we show Training Express output for the SACON knowledge base using typescript format, rather than the window-menu system provided for the student. In this example, the selection of the case file and other text provided by the teacher, which would normally be seen by a student, is not shown. Student input is in **bold face**. Teaching program output is in *italics*. Other text is output by the expert system.

M.1> teach sacon
Welcome to Training Express, the tutor for M.1 knowledge bases.

Loading the knowledge base...done.
Loading the secondary knowledge base...done.

To run the tutor, use the "go" command.

M.1> go
What is the name of the structure you wish to analyze?
>> 747-wing
Assuming that your characterization of the 747-wing in terms of its geometry, material characteristics, and boundary conditions are accurate, how much error (in percent) is tolerable for the analysis?
(Enter a number between 5 and 30.)
>> 5
Do you want to examine the responses of the 747-wing, evaluate its instability, or both?
>> examine

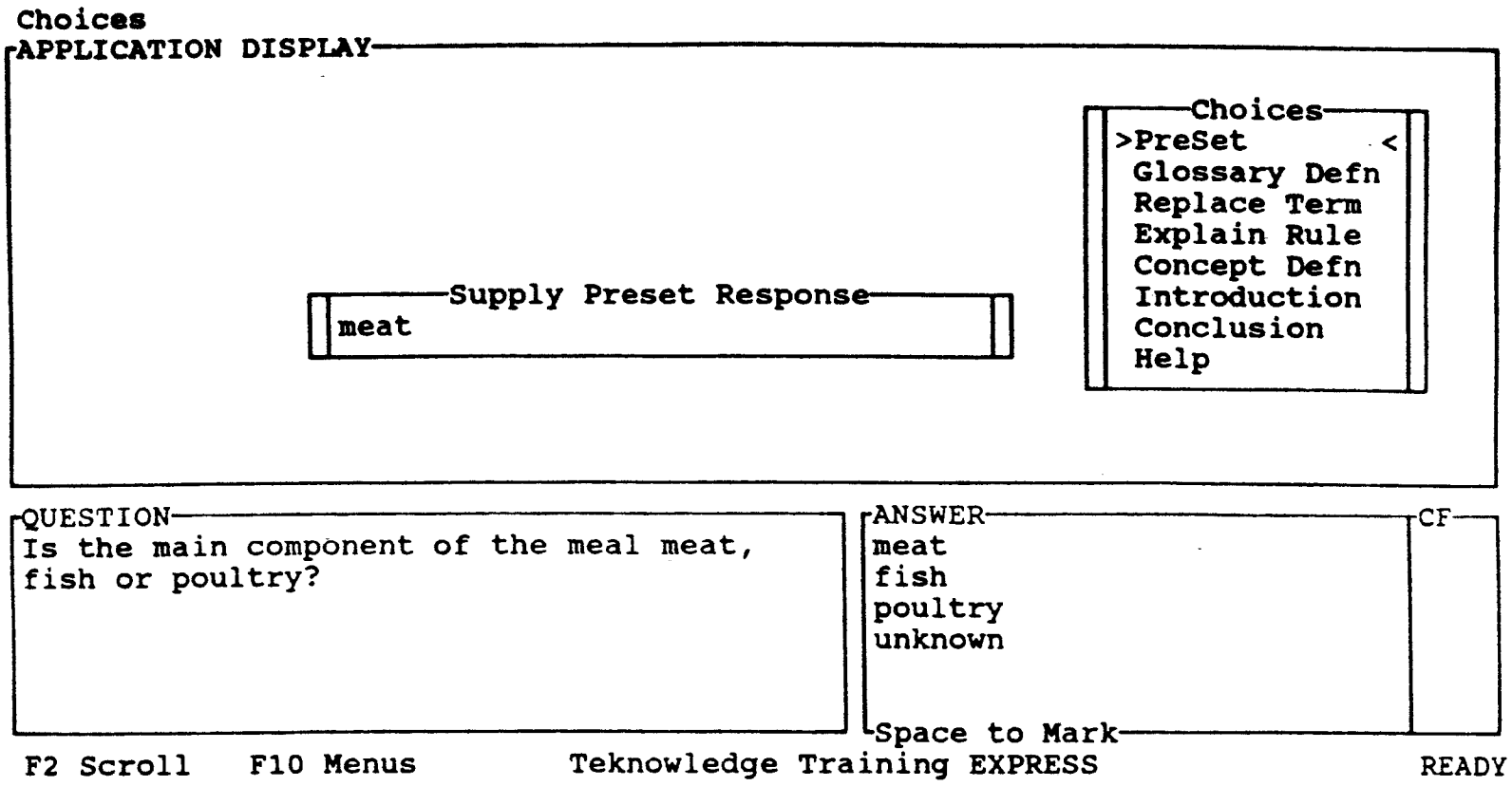

Figure 2-2: Defining a lesson by presetting case data

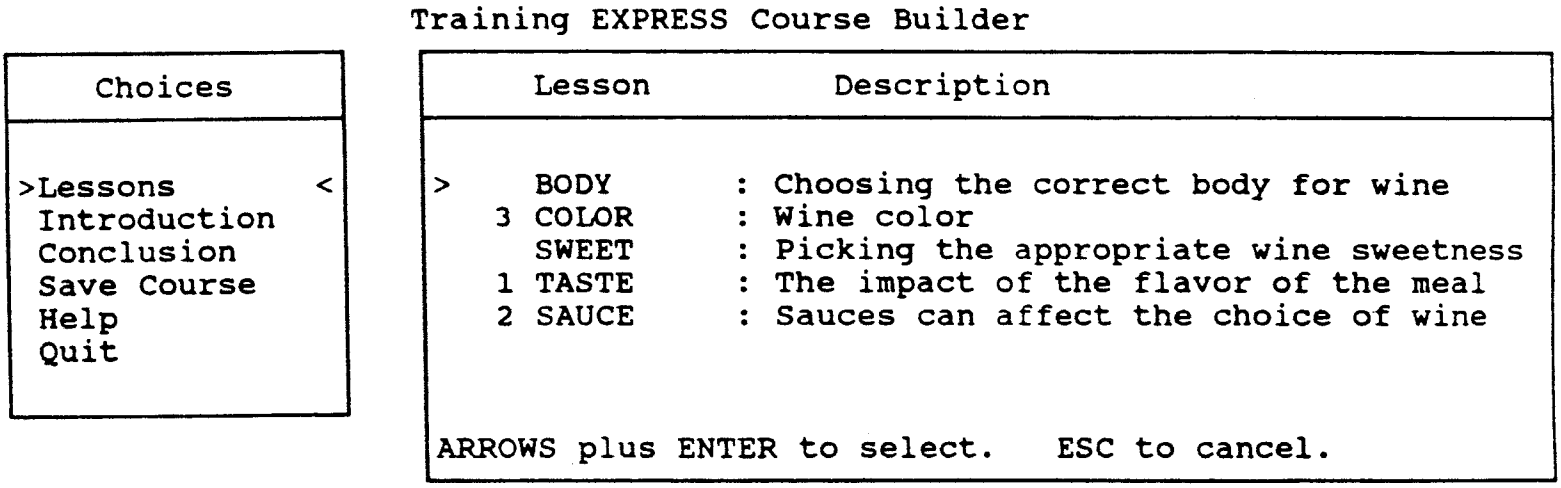

Figure 2-3: Defining a curriculum by ordering lessons

Does the 747-wing have any time dependent terms in its equations of equilibrium?
>> no
Is heating or cooling of interest in determining the structural responses of the 747-wing?
>> no
What are the names of the substructures of the 747-wing?
>> total-wing
What is the material composing most of the total-wing?
>> high-strength-aluminum
Enter the average longest dimension of the total-wing in inches.
>> 1420
Enter the weight of the total-wing in pounds.
>> 46000
Is the total-wing thin-walled or solid?
>> thin-walled
What is the configuration of support for the total-wing?
>> one-side
Is the total-wing to be modelled as a 1, 2, or 3 dimensional system?
>> 2
Would you describe the construction of the total-wing to be a
• network. (truss and/or frame elements)
• continuum. (sheets and/or blocks of material), or
• network and continuum. (a combination)?
>> continuum
Is the overall geometry of the total-wing curved or planar?
>> curved

------- Tutoring Interruption -------
What do you conclude about the shape of the total wing?
1. beam
2. plate
3. shell
>> **2**
No, the expert didn't conclude that the shape of the total wing is plate.
You should have concluded that the shape of the total wing is shell.
Which factors support the choice of shell?
1. construction of the total wing
2. modelling←dimensionality of the total wing
3. geometry of the total wing
>> **2**
Correct.

rule-56 mentions construction of the total wing and geometry of the total wing.
What geometry of the total wing is consistent with shell?
>> **options**
Your response must be chosen from the following:
curved, planar.
What geometry of the total wing is consistent with shell?
>> **planar**
Incorrect: planar.
rule-56 mentions curved.

------- Returning to Consultation -------
What are the names of the loadings on the total-wing?
>> flight
Enter the number of cycles of flight that will be applied.
>> 20000.0
How many components are there to flight?

```
>> 1
Would you describe flight-component-1 as being distributed over most of the
substructure or as acting at a point of the substructure?
>> distributed
Which surface of the substructure does flight-component-1 act normal to?
>> width-length
Enter the average wall thickness of the total-wing in inches.
>> .31
Enter the average depth of the total-wing in inches.
>> 41
Enter the magnitude of the distributed load (in psi).
>> 1
```

------- Tutoring Interruption -------
The expert concludes that the substructure nonlinearity of the total wing is material.

Which factors support the choice of material?
1. *errorclass*
2. *non-dimensional stress of the total wing*

>> **1,2**
Correct.

------- Returning to Consultation -------
Do the supports of the 747-wing involve Coulomb friction, nonlinear springs, and/or gapping?
>> **no**

------- Tutoring Interruption -------
What do you conclude about the ss←deflection of the total wing?
1. *kinematic←collapse*
2. *excessive←deflection*
3. *incremental←strain←failure*

>> 3
Good, the expert concluded that the ss←deflection of the total wing is incremental←strain←failure.
You should also have concluded that the ss←deflection of the total wing is excessive←deflection.

Which factors support the choice of excessive←deflection?
1. *sname*
2. *integrity←goal(STRUCTURE)*
3. *non-dimensional deflection of the total wing*
4. *errorclass*
5. *non-dimensional stress of the total wing*
6. *loading of the total wing*
7. *cycles(L)*

>> **2,5,6**
Incorrect: integrity←goal(STRUCTURE), nd←stress of the total wing and loading of the total wing.
See rule-59.
rule-59 mentions errorclass and non-dimensional deflection of the total wing.

What errorclass is consistent with excessive←deflection?
>> **list rule-59**
rule-59:
if errorclass = moderate and
nd←deflection(SS) = ND and
ND>= 0.0111111

then ss←deflection(SS) = excessive←deflection.

What errorclass is consistent with excessive←deflection?
>> **moderate**
Good, that is the conclusion considered by the expert.
See: rule-59.

------- *Returning to Consultation* -------

<< Tutoring interruption omitted >>

------- *Tutoring Interruption* -------
What do you conclude about the analysis class of the 747 wing?
1. *linear←analysis*
2. *nonlinear←boundary←condition*
3. *inelastic←crack←growth*
4. *general←inelastic*

>> **4**
Good, the expert concluded that the analysis class of the 747 wing is general←inelastic.

------- *Returning to Consultation* -------

The following analysis classes are relevant to the analysis of your structure:

general←inelastic

* *

The following are specific analysis recommendations you should follow when performing the structure analysis:

Activate incremental stress-incremental strain analysis.

Model nonlinear stress-strain relations of the material.

Solution will be based on a mix of gradient and Newton methods.

Special code should be written to scan peak stress at each step and to evaluate fatigue.

<< Section Omitted >>

2.4. Discussion of the tutoring example

As can be seen, SACON runs in the normal way, using either a case preset by the teacher or a case chosen by the student. The "whenfound" declarations inserted by the authoring system cause the tutoring interruptions to occur. The teaching program is passed the name of a concept (M.1 expression). Under rule-based control, the teaching program then examines the

conclusion made for the current expression, probes the student, provides feedback, and probes at deeper levels when the student makes mistakes.

It is not the aim of this research to argue for particular probing strategies, but rather to demonstrate what is easily possible and plausibly useful. As indicated these probes are similar to what have appeared in several other programs, particularly GUIDON.

The current version of Training Express illustrates three types of probes:

1. Ask student to state a value for an expression:

   << What do you conclude about EXPRESSION? >>>
   e.g., "What do you conclude about the wine?"

   This probe occurs when the expression is determined ("whenfound"); this is a primary breakpoint set by the teacher.

2. Ask student to state expressions that support a value:

   << What factors support the choice of VALUE? >>
   e.g., "What factors support the choice of zinfandel?"

   This probe occurs when the student fails to mention a value with a certainty greater than 50 in the first probe. The question is asked for each such "strongly believed missing value."

3. Ask the student what values for an expression are consistent with the conclusion made:

   << What EXPRESSION is consistent with VALUE? >>
   e.g., "What recommended-color is consistent with zinfandel?"

   This probe occurs when the student fails to mention a factor in the second probe. The question is asked for each "missing factor."

The pedagogical strategy used here prompts the student to use his knowledge to draw conclusions. When the student leaves something out, the program gives him the answer and checks if he can now remember why it is correct. A type 2 probe requests the concepts that are related to a conclusion. A type 3 probe more specifically asks the student to recollect the rule that relates two concepts by requesting the relevant value. As a whole, this approach engages the student in the ongoing consultation, giving him practice in applying general knowledge (a set of rules) to a particular problem (the case being discussed). It would also be

possible to ask the student to list relevant case data before the program requests it, thus going much further in drawing the student into the problem-solving activity.

2.5. Some Implementation Details

M.1 has several distinctive features that make it particularly suitable for instructional application:

- Both the knowledge base and cache of consultation conclusions are accessible and modifable under rule-based control, facilitating communication between the expert system and teaching program.

- Variables can be used freely, allowing rules to be written in general form. In particular, domain principles can be stated in general form, and teaching rules use variables to refer to domain concepts.

- External functions can be used to access a database and control the window-menu display.

The reasoning done by the teaching system is often complex. For example, consider the domain rule from SACON shown in Figure 4.

```
rule-56:
    if construction(SUBSTRUCTURE) is unique   id
       modelling←dimensionality(SUBSTRUCTURE) = MD and
       MD>= 2 and
       geometry(SUBSTRUCTURE) = curved
    then shape(SUBSTRUCTURE) = shell.
```

Figure 2-4: Domain rule from SACON encoded in M.1

The author shell extracts the "factors" from this rule:

```
rulefactors(rule-56) =
    [construction(SUBSTRUCTURE),
    modelling←dimensionality(SUBSTRUCTURE),
    geometry(SUBSTRUCTURE)].
```

The program knows that MD is a variable and which clause binds it. It knows that the rule concludes about an expression using a variable. Therefore, before discussing the rule with the student, the teaching program silently *reapplies* the rule and extracts the bindings for all of the variables. Thus, in the program's output we see "geometry of the total wing." This has been generated from "geometry(total-wing)" from a template supplied by the teacher during the authoring phase.

Much more complicated analysis is possible. For example, the program can determine that a rule clause is irrelevant because it sets a variable that is not needed for matching a domain fact. For example, if the fact "wine(red, ANY, dry) = zinfandel" is in the knowledge base, a clause setting a variable that matches "ANY" will be irrelevant to the successful application of the rule. The most difficult analysis involves determining the possible values a rule might conclude when a variable appears on the righthand side e.g., "if ... = W then wine = W."

As indicated, all analysis and teaching operations are carried out by M.1 rules. This is particularly advantageous for generating questions for the student and parsing his answers. In effect, the teaching program is carrying on a kind of "consultation," in which the probes are questions and the student's responses are data. All records of the student-teacher interaction are thus stored in M.1's cache and available for further reasoning by M.1 rules. In comparsion, GUIDON was implemented in a stylized version of Lisp, requiring time-consuming and complicated translation between EMYCIN's data structures and the list structures of GUIDON. Records of GUIDON's reasoning were stored in a completely different representation than MYCIN's consultation results, making the tutoring code much more complex and more difficult to maintain.

For the interested reader, Figure 5 is a typical rule from Training Express. It retrieves a rule from the knowledge base (using the kbentry primitive), determines the conjuncts of the premise (using other analysis rules), and maps over them, extracting the "expressions" (also determined by other analysis rules). Duplicates are removed from the resulting list. In all fairness, list manipulation is important in the tutor and such rules could not be successfully analyzed by the authoring program itself. So although the syntax is compatible, we don't quite have a teaching program that can teach about itself.

```
if listof(EXP,       kbentry(RULE:if PREM then ACT) and
                     conjuncts(PREM) = PROPS and
                     member(PROPS) = TERM and
                     expression(TERM) = EXP) = LST and
   dremdupls(LST) = SIMPLELST
then rulefactors(RULE) = SIMPLELST.
```

Figure 2-5: Typical Training Express rule written in M.1

3. Advantages and Limitations

The general pedagogical approach used is that of a case-method tutor. That is, the program teaches by discussing the application of general knowledge in specific situations. In many respects, this knowledge-based tutor has one of the simplest designs conceivable for a tutor of this type. The human teacher selects the concepts that will be discussed with the student. The case is prepared by the teacher or the student. In contrast, Guidon only works with canned

improvements. The output of Training Express makes clear the kind of teaching interaction clean rules allow, motivating and guiding the teacher (or his programmer assistant) in the process of rewriting the rules for the concepts he wants the program to discuss. The authoring system also provides suggestions about how rules can be rewritten using intermediate expressions that hide procedural computations.

Our experience also indicates that knowledge bases built specifically for teaching are stand-alone, small, and relatively easy to reconfigure. In practice a teacher is only likely to create a knowledge base good enough for the cases he wants to teach. In fact, the number of rules is not as important in many respects as the value of being able to show the student how actual problem cases are solved. Even a twenty rule system can reveal many subtle issues. The limitations of the system provide an excellent starting part for classroom discussion, and finding such limitations can be an important activity in learning from a knowledge-based tutor.

Thus, having the student adopt the attitude that the program is an object of study, that he is to evaluate the program, rather than vice versa, may mitigate some of the limitations of the approach. In particular, the apprenticeship method we follow doesn't challenge the student to to step through the entire problem himself. Interruptions occur while the expert system, not the student, is solving the problem. However, the teacher can encourage the student to vary the supplied cases and probe the program to print the rules or unwind the reasoning using M.1's WHY command. Thus, following the curriculum might be just a means a providing background to the more important activity of defeating the program and explaining why it fails.

The basic idea introduced here of having an authoring program help a teacher configure a knowledge-based teaching program can be applied to other forms of knowledge-based tutoring. For example, a simulation knowledge base, embodying a function-structure model of some process, might be analyzed in a similar way to point out difficulties the general teaching program will encounter or prompt the teacher to supply textual explanations for the student.

In one sense, the analysis program is acting as a student, looking over the knowledge base and complaining about constructs that are difficult to understand. Like other CAI authoring systems, the analysis program performs some of the tasks of a knowledge acquisition program. Indeed, Training Express could be used directly as a means of familiarizing experts with the capabilities of an existing expert system. The knowledge engineer could set breakpoints that will query the expert to fill in gaps in the knowledge base. Then, rather than comparing the expert's conclusions, factors, and values to the (non-existing) rules in the program, an augmented Training Express could write rules as the expert justifies his conclusions. This

demonstrates again the striking promise of knowledge-based programs, as mentioned in the opening paragraph of this paper. In particular, it shows the value of programs that can reason about other programs, illustrated well by Training Express' ability to write M.1 code that will interrupt an expert system, determine how variables are set and used, and even rewrite domain rules before they are presented to a student.

4. Status and Future Work

Training Express has been used with approximately a dozen M.1 knowledge bases, in domains ranging from patent law to structural analysis and medicine. The program has been released to selected customers for testing. The perceived value of the program appears to be very sensitive to the familiarity of the audience with the knowledge base and with the complexity of the knowledge base. That is, the program is most suitable for students who are already familiar with the domain of discourse of the expert system. The program is most suitable for knowledge bases with a high proportion of heuristics and facts, relative to procedural code (e.g., for conveying text to the user). Industrial R&D researchers were particularly interested in using the program during the test and verification cycle for M.1 knowledge base developers, in the manner described above.

Knowledge engineers also find that Training Express has considerable value over a simple WHY and HOW explanation facility. In particular, Training Express allows the developer to trap expressions that are always inferred by the system, but never presented to the user in the expert systems's output. In addition, Training Express can present glossary, concept and rule explanations when a break occurs. Currently M.1 and most rule-based systems only provide explanations when a question is asked.

Development of Training Express is continuing with users who wish to construct knowledge bases primarily for teaching, rather than adapting large existing programs, which poses more difficulties. Limitations such as those involving list manipulation will be handled as the need arises.

5. Conclusions

Training Express is a knowledge-based tutoring system that is designed to exploit the essential benefits of expert systems research in a classroom or on-the-job setting. We seek a trade-off, building on the advantages of a simple, rule-based design, and skirting the disadvantages by shifting some of the decision to a human teacher. In particular, we exploit the advantage of an expert system as a model that can be inspected and reasoned about by a teaching program. We separate the teaching program from the domain knowledge so teaching strategies are stated in a general way and easily reused. The breakpoint design focuses the

student interaction on the essential reasoning points, skirting messy code or unimportant details. Finally, the teacher is given a convenient facility for entering a library of cases, which combined with the breakpoints demonstrate the domain heuristics and facts she wishes to convey. We believe that this combination of features--interactive probing, situated (in context) learning, and sequenced examples--represents in large part the essential advantages of knowledge-based tutoring.

We believe that our design is a good example of the 80/20 rule: From 20% of effort involved in designing programs like NEOMYCIN and GUIDON, we believe we have derived 80% of the educational benefit. In many respects, Training Express is a case study in how academic research ideals can be adapted to application realities. Many trade-offs were made in moving from GUIDON to Training Express--

to make the program more reliable,
to build on well-known advantages of more traditional approaches
(e.g., the window-menu system for defining cases
and creating a curriculum), and
to provide a program that non-AI specialists could use today.

Indeed, our collaboration has revealed that there is an often unrealized middle ground between state-of-the-art research and commercial engineering. We didn't just take an existing research design (such as GUIDON) and "apply" it. We radically redefined the nature of the entire interaction between student and program. We brought the human teacher into the process and gave her an essential role. We developed a powerful authoring program that can relate rules and facts, coping with difficult problems involving variables. *Then* we "applied" the idea, packaging it with conventional hardware (e.g., personal computers) and software (the "C" programming language), and integrating it to a conventional database and window management system.

We found that producing a commercial product provides an entirely new orientation to research. In some respects, this is the well-known distinction between basic and applied research. We were faced with the constraints of a target audience and providing obvious value over current techniques. We weren't interested in just developing a theory or speculating on new instructional methods. We believe that as hardware makes delivery of complicated AI-based systems commonplace, many other researchers will want to engage in the same kind of collaboration, producing good research that anticipates use by actual teachers and students.

References

Bahill, A. T. and Ferrell, W. R. . *An introductory course in expert systems.* Technical Report, Systems and Industrial Engineering, University of Arizona, January 1986.

Bennett, J. S. and Engelmore, R. S. *SACON: A Knowledge-based Consultant for Structural Analysis*, pages 47-49, Proceedings of the Sixth International Joint Conference on Artificial Intelligence-79, Tokyo, Japan, August, 1979.

Clancey, WJ. From Guidon to Neomycin and Heracles in twenty short lessons (ONR Final Report 1979-1985). *The AI Magazine*, August 1986, *7(3)*, 40-60.

Clancey, W. J. *Knowledge-Based Tutoring: The Guidon Program.* Cambridge, MA: MIT Press 1987.

Collins, A., Brown, J.S., and Newman, S.E. *Cognitive Apprenticeship: Teaching the craft of reading, writing, and mathematics.*. BBN Technical Report 6459, Bolt, Beranek, and Newman, 1986.

Sleeman, D. and Brown, J. S. {editors.} *Intelligent Tutoring Systems.* New York: Academic Press 1982.