Representing Problem Solving for Tutoring and Critiquing Medical Domains

Kathy A. Johnson, Todd R. Johnson, and Jack W. Smith, Jr.

Laboratory for Knowledge-Based Medical Systems
Division of Medical Informatics
The Ohio State University
571 Health Sciences Library
376 W. 10th Ave.
Columbus, Ohio 43210

Email: kj@med.ohio-state.edu, tj@med.ohio-state.edu, smith.30@magnus.ohio-state.edu

1 Introduction

Medical problem solving often involves large amounts of data. This poses problems for experts solving the problem as well as for students learning to do so. There are often many different ways to solve a single problem. We have been studying the construction of tutoring systems for problems involving the interpretation of data. A system was built to detect errors in solving the problem of red blood cell antibody identification. The system has at its core a monitoring shell and a representation of problem solving. The shell provides a general method for monitoring that can be used in any domain for which an appropriate representation can be built. The shell and representation could also be used as part of a critiquing system to aid medical problem solving.

2 Representing Tutoring Knowledge

Many representations for tutoring have been explored as well as techniques for tutoring various domains. Given that a student has had some basic training in a problem solving area, tutoring can be viewed as a process during which a system must 1) monitor a student for mistakes, 2) determine the cause of the mistake, and 3) give the student appropriate information to correct his mistake. Monitoring for mistakes may be as simple as checking the final answer of an arithmetic problem or as complicated as detecting an incorrect sequence of steps during medical diagnosis. An example of a medical tutoring system taking this view of tutoring is Clancey's GUIDON system which uses the knowledge base from MYCIN to tutor students in the method for doing medical diagnosis (Clancey, 1987). Other medical systems have approached tutoring in a different manner best characterized by the term *critiquing*. Rather than correcting a student's problem-solving method, these systems present relevant information and alternatives to the student. Miller's VQ-Attending is an example of this type of system (Miller, 1986).

For either tutoring strategy, it is important to understand the problem solving taking place. This is obvious for a system that is trying to teach problem-solving strategy. However, a critiquing system must also know something about the problem solving in order to present relevant information. In order to filter all the information available to aid problem solving, the system needs know what goals the information will be serving.

We have been studying the domain of red blood cell antibody identification which is a multi-step problem-solving process similar to medical diagnosis. Data from many sources must be incorporated to determine the best answer as to the patient's antibodies. There is a great deal of variability in correct problem solving behavior. The student should be allowed to pursue any correct problem-solving path. In order to build such a system, we needed a representation of problem solving that is capable of encoding all possible correct problem-solving paths.

One of the most difficult aspects of describing flexible problem solving is finding a way to easily represent alternative methods to solve a problem. The standard representation for problem solving is a procedural description in which each operator is listed in order along with conditionals and loops. Such descriptions are notoriously inflexible and brittle (Newell, 1990). The ordering of the operators must be completely specified and the justification for choosing one operator before another is lost in the encoding. An alternative representation for problem-solving methods is embodied in the problem space computation model (PSCM) of Newell (Newell, 1990). In the PSCM, all problem solving is viewed as search for a goal state in a problem space. The problem space contains a set of operators which when applied in the appropriate sequence will achieve the goal. Knowledge about when particular operators are applicable to a state can be specified independent of knowledge about which operator to prefer. For example, conditions may be appropriate for moving Block X or Block Y (e.g. both are clear). Other kinds of knowledge may indicate that it makes no difference which is moved first, or perhaps that it is better to move Block X first. Operators may be encoded directly or they may themselves be set up as a goal in a sub-problem space with their own set of operators and control knowledge. For example, moving a block may be a primitive operator tied to a motor system while clearing a block might require a subgoal to determine the sequence of operators needed to remove the block on top

* This research is supported by National Heart Lung and Blood Institute grant HL-38776, and National Library of Medicine grant LM-04298.

Goal: Identify-Allo-Antibodies
If: not auto-antibodies, desired allo-antibodies identified
To-Make: Complete, Consistent, Parsimonious explanation for data
By: Method1

Method: Method1
Goals: Make-abstract-hypotheses, Match-hypothesis-to-antigram, Rule-out, Explain *datum*
Control: Rule-out is better than Explain *datum*
Other operators are indifferent

Goal: Make-abstract-hypotheses
If: *datum* needs to be explained
To-Make: antibody *a* explains *datum*
By: Group-by-rows, Group-by-columns, Group-by-like-reactions
Control: Group-by-rows, Group-by-columns, and Group-by-like-reactions are indifferent.

Goal: Match-hypothesis-to-antigram
If: antibody <a> does not have a specificity
To-Make: antibody <a> has a specificity
By: Method3

Goal: Rule-Out
If: desired limit-possible-explanations
To-Make: antibody <b> ruled-out (i.e. not considered as part of the explanation)
By: Method4

Goal: Explain datum
If: *datum* needs to be explained
To-Make: antibody *a* explains *datum*
By: Method5

Figure 1: Partial model of goals and methods for red blood cell antibody identification

of it. A description of problem solving in these terms can provide a more dynamic or flexible description than does an algorithm because operator proposal knowledge and search-control knowledge can be sensitive to the current state of problem solving. In addition, alternative methods for achieving goals can be represented as different sets of subgoals and knowledge to choose between alternative methods may or may not be provided.

The PSCM allows us to describe flexible problem-solving methods that we need for monitoring a student's problem solving. To make use of this knowledge, however, we need to explicitly represent what each problem space and operator does. We have chosen to represent that knowledge using terminology borrowed from the functional representation developed at Ohio State (Sembugamoorthy and Chandrasekaran, 1986). Previously, the functional representation had been used to describe devices, plans, and computer programs (Allemang, 1990; Chandrasekaran, et al., 1986; Keuneke, 1989; Sticklen, 1987). The traditional functional representation consists of a function, what result is expected, what conditions must be true before the function will be achieved, and the behavior by which the function is achieved. The behavior is a sequence of state transitions connected by subfunctions. In our case, a function is a goal to be achieved, the result is the state of problem solving after achieving the goal, there are still preconditions, and we have a method (or methods) for achieving the goal instead of a specific device behavior. The difference between a device behavior and a problem-solving method is in the variability of the steps. A device usually behaves in a deterministic manner since a device is a physical system that works in a linear fashion (seldom do devices randomly execute functions unless they are designed to do so). The method for problem solving should allow for variability in sequence and is represented as a set of subgoals with whatever control knowledge is necessary (or desired).

3 Monitoring Students Using a Functional Representation of Problem Solving

The functional representation of problem solving for red blood cell antibody identification was extracted from a task analysis that was used to develop an expert system (Johnson, et. al. 1991). The task analysis was facilitated by the use of problem spaces describing generic tasks (Johnson, 1991). A portion of the functional representation is shown in Figure 1. A monitoring system (Johnson, 1993) using this functional representation has been built in Soar. It is capable of mapping a simulated student's intermediate results into the goal that he was pursuing and determining if there is an error in the student's reasoning[1]. Figure 2 shows the interactions of the components of the monitoring shell.

3.1 Monitoring Goals

The following are the goals that comprise the monitoring task:

- *Get-student-result*: Get the student's intermediate conclusion.
- *Generate-explanations-for-result*: Determine the goals that the student could have been pursuing that would generate such a result
- *Determine-possible-goals*: Determine the goals that are possible from the previous goal that generate visible results.
- *Evaluate-validity-of-goal:* For a particular goal generated by *generate-explanations-for-result*, determine whether the goal was valid by checking the preconditions and search-control.

1. The current monitor is in a version of Soar written in Lisp and is too slow to be used with actual students. Additionally, a user interface needs to be developed to capture the student's results. The data used to test the monitor was representative of student's actual results.

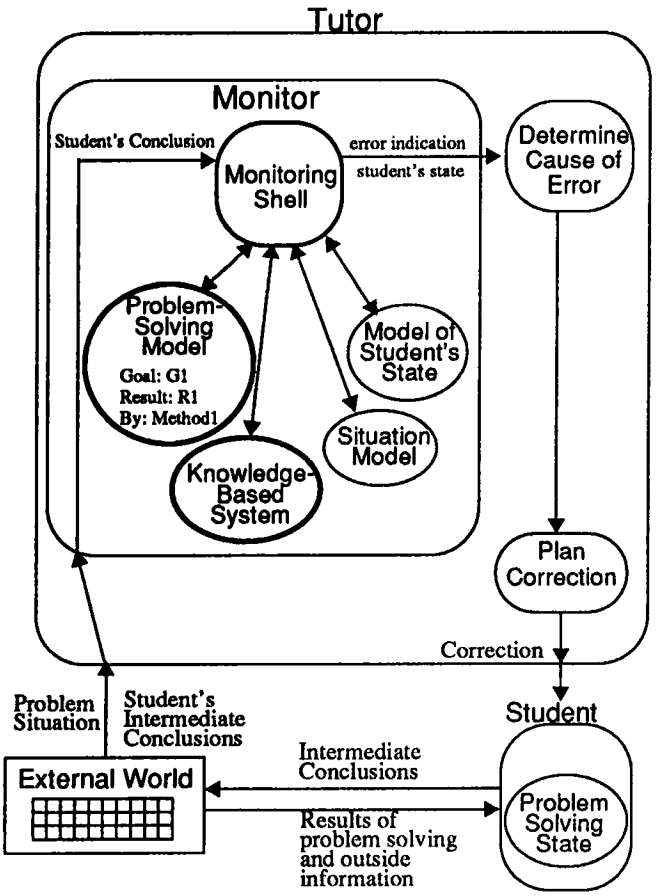

Figure 2: The high level problem-solving breakdown of the tutor and the monitor

- *Determine-explanation*: Of the valid explanations (goals) generated, determine which fits the student best.
- *Evaluate-validity-of-method*: Determine whether the conclusion was correct for the goal being pursued. This is generally accomplished by running the appropriate portion of the expert system and comparing the result to the student's intermediate conclusion.

These are general goals that rely on having an explicit functional model of problem solving and some way of checking the correctness of a conclusion. That is, they are useful in any system that has those properties. They are incorporated into the expert systems and are used when the system has a goal of *monitor-student-for-error* rather than the expert's usual domain goal such as *identify-allo-antibodies* in the case of red blood cell antibody identification. *Get-student-result* has domain-specific parts that depend on the particular information available such as menu selections on a screen, mouse clicks in active areas, text input, etc. Likewise, *evaluate-validity-of-method* needs to have access to a knowledge-based system in the domain. When the monitoring shell detects an error, it is noted in its state and the model of the student's state is available for further processing.

The general method for composing these goals is:

1. The system scans the problem-solving situation that is given to the student to get a preliminary model of the situation.
2. *Get-student-result* is set as a goal and it returns the student's intermediate conclusion.
3. *Generate-explanations-for-result* is used to determine the goals that the student could have been pursuing.
4. *Determine-applicable-goals* generates a set of goals that are appropriate in the current situation by finding goals whose preconditions match. Search control is not checked for these goals, it is deferred to *evaluate-validity-of-goal.*
5. *Evaluate-validity-of-goal* is used for each goal in the set of goals returned from *generate-explanations-for-result*. If the goal being tested does not appear in the set of applicable goals generated by *determine-applicable-goals*, then the goal is marked invalid and a reason of not-applicable is listed. If there is a match, then search control is checked. If the search control indicates that the goal is not the best one at this time, then the goal is marked invalid and a reason of wrong-time is listed. If the search control checks out, then the goal is marked as valid.
6. *Determine-explanation* is used after all the goals have been checked for validity. If no goal is valid, then an error is determined and the set of goals and reasons for them being invalid is made available for further error diagnosis. If one goal is valid, then it is assumed that the student is pursuing this goal even if there were other invalid goals. If more than one goal is valid, then one is picked at random and the others are kept in the student model as alternative goals.
7. *Evaluate-validity-of-method* is then used to check the actual value of the student's conclusion to make sure that it matches the correct value. At this point, the expert system is run to find the correct value. If it does not match, then an error is determined. If it matches then the student model is updated with the conclusion.

If no errors were indicated, then the process begins again from step 2.

3.2 Classes of error

There are several classes of error that the monitoring shell detects.

1. Wrong timing—The student may be pursuing a goal that is applicable, but is not the best one at this time. That is, the preconditions for the goal were met, but

search-control indicated that there was something else that should be done first.

2. Goal not applicable—The student was attempting a goal whose preconditions were not met.
3. Goal not part of current method—The student was attempting a goal not in the current method. Whether the new goal's preconditions were met or not, this is an error because the current goal hasn't been achieved yet.
4. Bad implementation of goal—The student was attempting a correct goal, but the result of executing the goal was not correct.

It is necessary to detect these types of errors in any domain. Each error type will have domain-specific explanations. There are also process-specific errors and explanations for reasoning strategies such as abductive assembly. These errors and explanations would be useful in any domain employing the method of abductive assembly.

4 Critiquing Using a Functional Representation of Problem Solving

We have shown how a functional representation of problem solving can be used to monitor a student for errors (an essential subtask of tutoring). The same monitor may be used in the context of a critiquing system. A critiquing system provides a person with support for problem solving. It may indicate possible omissions of steps and inconsistencies in conclusions. It can provide access to descriptions of alternative methods to achieve a particular step. It can also help to organize the information presented to a person. These functions are possible only if the system is able to determine the goal of the person solving the problem.

The monitoring shell and functional representation could be used in a critiquing system as follows. The monitoring shell infers the problem solver's goal. If the problem solver has omitted a step, the monitor detects this and puts up a warning. As the person solves the problem, the monitor also compares the person's answer to its expert system answer. If these are found to be significantly different, the system warns the person. Since the critiquing system contains a description of the problem solving organized around goals and methods, it has easy access to a list of methods. These could be made available upon request. Finally, the problem-solving methods in the system make reference to certain types of data. Enhancement of the functional representation would provide a goal-oriented index to the relevant data. This could then be used to filter the data for display to the human problem solver.

5 Conclusion

Medical problem solving is a complex process often involving large amounts of data. Solving such problems creates difficulties for experts solving the problem as well as for students learning to do so. We have built a shell for detecting errors in problem solving. This shell may be used at the core of a tutoring system or a critiquing system. It provides the capability to follow a problem-solving path in a flexible manner. The problem-solving representation that is a key part of the shell provides a goal-oriented index of methods and could be expanded to also index data relevancy. Future work will explore both the tutoring and critiquing aspects of the monitoring shell.

Acknowledgments

We thank the AIM group for their assistance and comments on this paper and the work it reports. We also thank the members of the Soar community for their intellectual and technical support. This research is supported by National Heart Lung and Blood Institute grant HL-38776, and National Library of Medicine grant LM-04298.

References

Allemang, D. T. (1990) *Understanding Programs as Devices*. Ph.D., The Ohio State University.

Chandrasekaran, B. (1986). Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design. *IEEE Expert, 1*(3), 23-30.

Chandrasekaran, B., Josephson, J., and Keuneke, A. (1986). Functional Representations as a Basis for Generating Explanations. In *Proceedings of the 1986 IEEE International Conference on Systems, Man and Cybernetics* (pp. 726-731). Atlanta, Georgia: IEEE Computer Society Press.

Clancey, W. J. (1987). *Knowledge-Based Tutoring; The Guidon Program*. MIT Press.

Johnson, K. (1993) *Exploiting a Functional Representation of Problem Solving for Error Detection in Tutoring*. Ph.D., OSU.

Johnson, K., Johnson, T., Smith, J. W., DeJongh, M., Fischer, O., Amra, N., and Bayazitoglu, A. (1991). Red-Soar—A System for Red Blood Cell Antibody Identification. In *Symposium on Computer Applications in Medical Care*. Washington, D.C.

Johnson, T. R. (1991) *Generic Tasks in the Problem-Space Paradigm: Building Flexible Knowledge Systems While Using Task-Level Constraints*. Ph.D., OSU.

Keuneke, A. (1989) *Machine Understanding of Devices: Causal Explanations of Diagnostic Conclusions*. Ph. D., The Ohio State University.

Laird, J. E., Newell, A. and Rosenbloom, P.S. (1987). SOAR: An Architecture for General Intelligence. *Artificial Intelligence*, Vol. 33(No. 1).

Newell, A., Yost., G., Laird, J.E., Rosenbloom, P.S. and Altmann, E. (1990 September 24-26). Formulating The Problem Space Computational Model. In *Proceedings of the 25th Anniversary Symposium, School of Computer Science, Carnegie Mellon University*. Pittsburgh, PA.

Miller, P. L. (1986) *Expert Critiquing Systems: Practice-Based Medical Consultation by Computer*. New York: Springer-Verlag.

Sembugamoorthy, V., and Chandrasekaran, B. (1986). Functional Representation of Devices and Compilation of Diagnostic Problem Solving Systems. In J. Kolodner & C. Reisbeck (Eds.), *Experience, Memory and Reasoning* (pp. 47-73). Lawrence Erlbaum and Associates.

Sticklen, J. (1987) *MDX2: An Integrated Medical Diagnostic System*. Ph.D. OSU.