

PROJECT ADMINISTRATION DATA SHEET

ORIGINAL  REVISION NO. \_\_\_\_\_

Project No. G-36-616 GTRI/~~GTR~~ DATE 7 / 12 / 84

Project Director: Janet L. Kolodner School/~~ICS~~ ICS

Sponsor: National Science Foundation

Type Agreement: Grant No. IST-8317711

Award Period: From 6/15/84 To 11/30/87 (Performance) 2/28/88 \*\* (Reports)

Sponsor Amount: This Change Total to Date

Estimated: \$ \_\_\_\_\_ \$ 210,000

Funded: \$ \_\_\_\_\_ \$ 210,000

Cost Sharing Amount: \$ 11,054 Cost Sharing No: G-36-350

Title: "Extracting Information from Experience: Experience Driven Incremental Learning"

ADMINISTRATIVE DATA

OCA Contact Lynn Boyd x4820

1) Sponsor Technical Contact:

2) Sponsor Admin/Contractual Matters:

Ronald R. Yager

Stephen G. Burnisky

National Science Foundation

Grants Official

Division of Information Science & Tech.

National Science Foundation

Washington, DC 20550

Washington, DC 20550

(202) 357-9569

(202) 357-9671

Defense Priority Rating: n/a

Military Security Classification: n/a

(or) Company/Industrial Proprietary: n/a

RESTRICTIONS

See Attached NSF Supplemental Information Sheet for Additional Requirements.

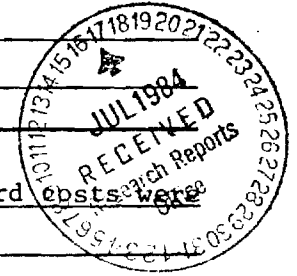
Travel: Foreign travel must have prior approval -- Contact OCA in each case. Domestic travel requires sponsor approval where total will exceed greater of \$500 or 125% of approved proposal budget category.

Equipment: Title vests with GIT.

COMMENTS:

\*Advance Project No. was assigned in the amount of \$13,868. Also, preaward costs were approved on OPAS Form. (See OPAS form and Advance Project No. Request.)

\*\*Includes usual 6 month unfunded flexibility period.



COPIES TO:

Sponsor I.D. #02.107.000.84.007

Project Director  
Research Administrative Network  
Research Property Management  
Accounting

Procurement/EES Supply Services  
Research Security Services  
Reports Coordinator (OCA)  
Research Communications (2)

GTRI  
Library  
Project File  
Other Newton

SPONSORED PROJECT TERMINATION/CLOSEOUT SHEET

Date 6/17/88

Project No. G-36-616 School/Lab ICS

Includes Subproject No.(s) N/A

Project Director(s) J. L. Kolodner

GTRC/GIT

Sponsor National Science Foundation

Title Extracting Information from Experience: Experience Driven Incremental Learning

Effective Completion Date: 11/30/87 (Performance) 2/28/88 (Reports)

Grant/Contract Closeout Actions Remaining:

- None
- Final Invoice or Copy of Last Invoice Serving as Final
- Release and Assignment
- Final Report of Inventions and/or Subcontract:  
Patent and Subcontract Questionnaire sent to Project Director
- Govt. Property Inventory & Related Certificate
- Classified Material Certificate
- Other \_\_\_\_\_

Continues Project No. \_\_\_\_\_ Continued by Project No. \_\_\_\_\_

COPIES TO:

Project Director  
 Research Administrative Network  
 Research Property Management  
 Accounting  
 Procurement/GTRI Supply Services  
 Research Security Services  
 Reports Coordinator (OCA)  
 Program Administration Division  
 Contract Support Division

Facilities Management - ERB  
 Library  
 GTRC  
 Project File  
 Other \_\_\_\_\_  
 \_\_\_\_\_  
 \_\_\_\_\_



GEORGIA INSTITUTE OF TECHNOLOGY  
SCHOOL OF INFORMATION AND COMPUTER SCIENCE • ATLANTA, GEORGIA 30332 • (404) 894-3152

January 30, 1986

Dr. Beth Adelson  
Division of Information Science and Technology  
National Science Foundation  
Washington, DC

RE: NSF Grant No. IST-831171, Extracting Information from Experience: Experience-Driven Incremental Learning

Dear Beth,

This letter will serve as the first end-of-year report for my grant entitled: Extracting Information from Experience: Experience-Driven Incremental Learning. I am sorry to be sending it so late.

In this project, we study how a problem solver can extract information from experience, as a result becoming more expert at its reasoning task. The problem is being addressed in the framework of several reasoning systems. As a result of experience, knowledge structures are modified and new structures are created, resulting in a memory that integrates factual and experientially-acquired knowledge. Three methods of learning are being investigated: generalization, failure-driven explanation, and integration of new items into memory for later analogy. The method of problem solving that we are investigating is case-based reasoning. In case-based reasoning, the results of previous cases are brought to bear in solving new problems. This can result in two efficiencies in problem solving: past problem solving errors can be avoided and shortcuts in problem solving can be used.

Our research began with an introspective investigation in the domain of psychiatric diagnosis and treatment. Working together with a psychiatrist and examining in detail the reasoning he used in several of his difficult cases, we were able to discover several of the roles experience plays in problem solving (Kolodner & Kolodner, 1982) and to devise an algorithm for diagnosis that includes recourse to previous experience as one of its elements (Kolodner & Kolodner, 1985). Our program, called SHRINK, is able to distinguish between several affective disorders for some "easy" cases. One of its methods is to be reminded of previous cases and to focus its consideration of patient symptoms on what had been important for diagnosis in a previous case.

Based on the initial investigation into diagnosis and treatment, we have derived a framework that explicitly states the relationships between experience and problem solving. Our framework, which we call *natural problem solving*, integrates problem solving, learning, and analogy. We are investigating and refining that framework in the context of several computer problem solving systems that use case-based reasoning. We provide a synopsis of the year's work in the next few paragraphs. Additional detail is presented in attached papers, especially (Kolodner & Kolodner, 1985) and (Kolodner, 1985).

We call the kind of problem solving that makes use of experience *natural problem solving* (Kolodner, 1985). Natural problem solving integrates learning and analogy with problem solving itself. Learning occurs

as a natural consequence of problem solving. If a novel procedure is derived in the course of solving a complex problem and all goes well in its execution, then a new procedure is learned for dealing with this new class of problems. If problems are encountered while applying known knowledge to a new case, the results of analyzing and fixing the problems lead to refinement or modification of problem solving knowledge. Because general knowledge is not always adequate to deal with novel situations, analogy to previous similar novel situations is necessary to deal with their complexities. Thus, remembering a previous case to use in later problem solving is one of the necessary learning processes. Because descriptions of problems may be incomplete, problem understanding is a necessary prerequisite to coming up with a solution and is part of the natural problem solving framework. Feedback and analysis of feedback through follow-up procedures are also part of natural problem solving. Without evaluation processes based on feedback, learning could not happen and analogy to previous experience would be unreliable.

Within that framework, our approach has two components (Kolodner & Simpson, 1985) *experience-driven learning* and *case-based problem solving*. We wrote extensively about experience-driven learning in our proposal. In experience-driven learning, *experience contributes to refinement and modification of reasoning processes and knowledge*. Experience drives the learning process. One of the learning processes we identified previously, we called *similarity-driven generalization*. Our explanation of that process was that as similarities between experiences were seen, generalizations were built by extracting the similarities from the cases. Our recent investigation into experience-driven learning looks at the controls on this generalization process. In particular, we will claim that the problem solver can exert control over the learning process. As problem solving is happening, the new experience is added to the memory (Kolodner & Simpson, 1984). Similarities between cases lead the problem solver to use predictions from the previous case to resolve the new case (see later paragraph about case-based reasoning). While our previous model said that generalizations could be expected each time similarities were noticed, our current model stresses that a prediction made by a previous case should hold before a generalization is made. If predictions hold, then generalizations are drawn that will be useful in analyzing a future situation. These generalizations form an *explanation* (DeJong, 1983) that coherently ties the reasoning steps together.

Here we must explain how our explanations and explanatory process differ from DeJong's. He assumes that only one generalized explanation should be created based on one initial case. Any particular problem solving episode, however, may contain several explanatory predictions. In our formulation, generalizations are not made until the predictions from an old case are needed to solve a new problem. Then, only the part of the old case used to reason about the new one is generalized. Both the generalization and the case are then available for later reasoning: the generalization will be used when features of a new case correspond to those that are part of the explanation that was built; the case when different features of a new problem are shared with it. In this way, multiple explanatory generalizations can be created, but only when experience shows them to be potentially useful.

*Case-based reasoning* uses previous experiences to suggest means of solving new problems. Thus, *individual experiences act as exemplars upon which to base later decisions*. Recall of a previous experience can aid in understanding the intricacies and focus of a new problem, generating a plan for resolution of the problem, and in case of failure, in explaining and remedying the failure and re-evaluating the case. Case-based reasoning allows shortcuts in problem solving and avoidance of previous mistakes (Kolodner & Simpson, 1985; Simpson, 1985). More detail about these processes can be found in an attached paper (Kolodner, 1985).

We are exploring the framework presented above in several real-world domains, both common-sense and expert. Our MEDIATOR project [KOL85a] [SIMP85] resolves common sense disputes based on experience solving previous similar problems. By common-sense disputes, we refer to the kinds people run into from day to day. Children quarrelling over possession of objects, colleagues needing the same resource at the same time, and disputes encountered in reading the newspapers are just a few of the kinds of disputes the program deals with. The problem solving episode above is an example from the MEDIATOR. The MEDIATOR begins with a semantic memory detailing the kinds of disputes it might encounter (e.g., physical,

economic, political) and a set of common mediation plans (e.g., split in half, split the difference, divide by parts, share). As it resolves disputes, it builds up an episodic memory organized by the concepts it started with. During processing, it first attempts recourse to previous experience to resolve a problem, and if none is available it uses default means (based on consideration of all alternatives). It learns based on feedback about the decisions it has made. If feedback is positive, it reinforces its belief that a particular type of plan is appropriate to a particular type of problem by storing the case and the plan used to resolve it. When it encounters later problems with features similar to one it has stored in memory, it is reminded of that case and checks to see if the plan used there is appropriate to its new problem. A positive experience may thus provide a shortcut in later problem solving. If feedback is negative, the MEDIATOR tracks down its error, fixes the knowledge that was responsible and attempts resolution of the problem a second time based on the new knowledge learned during feedback and the corrected knowledge that caused the previous error. When it finally resolves the problem satisfactorily, and stores the entire case in memory, later reminding of that case will (1) allow the problem solver to resolve a later similar problem without making the same mistakes a second time or (2) help the problem solver to figure out what went wrong when a similar failure occurs in the future.

There are several novel aspects to the MEDIATOR project. First, its model of problem solving includes not only the planning part of problem solving, but also problem understanding and failure resolution based on feedback. Case-based reasoning can facilitate reasoning during any of these phases of problem solving. Second, the analogical transfer process is *demand driven*, where demand is provided by the task the problem solver is carrying out. When the problem solver is trying to understand a problem, it is the problem classifications (i.e., representational schemata) of previous cases that it investigates for applicability to the new problem. When it is attempting to derive a skeletal plan, it is the abstract plan from the previous case that it checks for applicability. Third, the MEDIATOR has a well-articulated long term memory for experience. A problem solving experience presented to the MEDIATOR is indexed in memory by those features that differentiate it from other similar experiences stored there. A fourth novel feature of the MEDIATOR is in its use of the same problem solving model to both solve domain problems (in this case, to resolve disputes) and to track down and fix failures in reasoning. It is able to do this because it treats both types of problems as first, classification problems, and then, plan instantiation problems.

One of our expert domains is mediation of labor/management disputes. Based on consultation with a labor mediator (Prof. Sherman Dallas), we have created a program, called the PERSUADER [SYC85a] [SYC85b], that uses precedent setting cases and prevailing practice, along with distinctive features of the particular case, to derive equitable contracts. The PERSUADER presents its suggestions to the parties involved, and based on their feedback, it can either modify its suggestions or derive an argument to persuade whichever party is in disagreement of the utility of the proposed contract. Like the MEDIATOR, the PERSUADER uses previous cases to create solutions to new problems. Unlike the MEDIATOR, it must deal with the multiplicity of goals of the two parties involved, including the stated goals as well as the face-saving and power-maintaining goals of the negotiators themselves. It must also consider outside influences such as the state of the economy, economic trends, industry trends, the company's financial situation, and the make-up of the union. It thus uses previous cases to come up with an initial contract suggestion, and before suggesting it, modifies it based on these outside influences. These modifications may also be based on previously-made contract modifications.

While concentration in the MEDIATOR is on the problem solving framework itself and the times during problem solving when case-based reasoning might be useful, concentration in the PERSUADER is on the problem of analogical transfer in a highly-complex domain. Both efforts are aimed toward investigating how more efficient problem solving systems can be designed and built.

Another of the problems we are beginning to address is the way in which a case-based problem solver ought to interact with the memory for cases. There are several ways that a case-based problem solver and a memory might interact. One way is for the problem solver to ask the memory at specified times in its processing whether any cases are available. This is the way our MEDIATOR and PERSUADER work. Under that framework, each time the problem solver needs to make a decision, it first asks the memory system for

relevant cases. This interaction seems to have the advantage of keeping memory processes tightly controlled. Only cases with features similar to the current problem the problem solver is working on are ever retrieved. On the other hand, this seems unrealistic with respect to natural memory processes: Memory returns only what it can come up with on the spot. It is never given the opportunity to "ponder" and contribute later (when the problem solver might be working on something else). In particular, it is impossible this way to capture the capability people have of remembering more about a previously-considered situation after considering aspects of another situation, and then using that "unrelated" reminding in their problem solving.

What we wish to develop is a process model containing co-operating but autonomous memory and problem solving processes. The problem solver keeps memory informed about what it is doing, and the memory uses the problem solver's focus to search its structures. When the problem solver needs knowledge to do some task, it asks memory. Memory informs the problem solver immediately of any generalized knowledge or cases that it can find easily and that are directly related to the problem solver's goal. The problem solver continues its computations with that knowledge, and at the same time, memory continues its search for relevant cases. As the problem solver continues its computing, it may change its focus or add new knowledge to the problem statement or solution. Memory uses that knowledge in two ways: to create new search specifications, and to update old search specifications. In this way, it has the capability of relating the problem solver's range of focusses to each other, coming up with and remembering novel combinations of plans and the objects they are applied to.

This architecture would allow us to build a problem solving system that can use its previous experience to suggest means of doing things that are different from but better than what is specified in the problem statement. Suppose, for example, an advisor is asked for recommendations on *buying* bookshelves. If its memory is well-organized and autonomous, and if it is able to use any combination of the set of cues available from the problem statement, then it might be able to come up with novel suggestions that have worked well previously. When money is limited, for example, it might suggest *building* bookshelves instead of buying them. If the asker is a student, it might go further and suggest using cinderblocks and boards (a classic student design). While a problem solver by itself might also be able to come up with such solutions, the key here is that the memory will be doing most of the work in changing the problem solver's mind, rather than the problem solver having to do a lot of work to discover a better solution than the one suggested in the problem statement.

Making memory autonomous gives it more control over what to look for. It can take its cues from a variety of places. While the problem solver can ask it for particular information it wants, memory can also consider cues from, e.g., the problem statement, previous reasoning attempts, recent conversations, or its own set of goals, and make its own contributions to the problem solving. Of course, the memory and problem solver have to be able to pass messages to each other, and their interactions must be controlled. We have begun to address some of these problems in our Consumer-Advisor program. One of the attached papers (Kolodner, 1985) has a bit about that program.

## **Publications and Presentations**

### **Publications**

- Kolodner, J. L. (1985). *Experiential Processes in Natural Problem Solving*, submitted to *Artificial Intelligence*. Also Technical Report No. GIT-ICS-85/23. School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA.
- Kolodner, J. L. & Kolodner, R. M. (1985). *Using Experience in Clinical Problem Solving: Introduction and Framework*, submitted for special issue of *IEEE Transactions on Systems, Man, and Cybernetics* devoted to causal and strategic issues in diagnostic problem solving. Also Technical Report No. GIT-ICS-85/21. School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA.

- Kolodner, J. L. & Simpson, R. L. (1986). Problem Solving and Dynamic Memory (long version). In Kolodner, J. L. & Riesbeck, C. K., *Experience, Memory, and Reasoning*, Lawrence Erlbaum Associates, Hillsdale, NJ. Also submitted for special issue of *Cognitive Science* devoted to memory, experience, and reasoning. Also Technical Report No. GIT-ICS-84/24. School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA.
- Kolodner, J. L., Simpson, R. L., & Sycara, E. (1985). A Process Model of Case-Based Reasoning in Problem Solving. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August, 1985, pp. 284-290. Also Technical Report No. GIT-ICS-85/01. School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA.
- Kolodner, J. L. (1985). Memory for Experience. In Bower, G., ed., *The Psychology of Learning and Motivation*, Academic Press, New York, 1985, pp. 1-57. Also Technical Report No. GIT-ICS-84/23. School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA.
- Kolodner, J. L. (1986). Using Experience as a Guide for Problem Solving, in *Machine Learning: A Guide to Current Research*, Kluwer Academic Press, 1986.
- Kolodner, J. L. (1985). 'If you want my advice...': Some protocols of a memory intensive task, in *Proceedings of the Second Annual Workshop on Theoretical Issues in Conceptual Information Processing*, New Haven, CT, May, 1985.
- Kolodner, J. L. & Simpson, R. L. (1985). Machine Learning Research at Georgia Tech: Using Experience as a Guide for Problem Solving, in *Proceedings of Machine Learning Workshop*, Skytop, PA, June 24-26, 1985, pp. 96-99.
- Kolodner, J. L. & Riesbeck, C. K. (1986). *Experience, Memory, and Reasoning*. (edited volume) Lawrence Erlbaum Associates, Inc., expected May, 1986.
- Sycara, Katia (1985). Precedent-Based Reasoning in Expert Labor Mediation. Technical Report #GIT-ICS-85/22. School of Information and Computer Science. Georgia Institute of Technology, Atlanta, GA.
- Sycara-Cyranski, Katia (1985). Arguments of Persuasion: Two Papers. Technical Report #GIT-ICS-85/19. School of Information and Computer Science. Georgia Institute of Technology, Atlanta, GA.
- Sycara-Cyranski, Katia (1985). Persuasive Argumentation in Resolution of Collective Bargaining Impasses. *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, Irvine, CA, August, 1985.
- Sycara-Cyranski, Katia (1985). Arguments of Persuasion in Labour Mediation. *Proceedings of IJCAI*, Los Angeles, CA, August, 1985.
- Simpson, R. L. (1985). A Computer Model of Case-Based Reasoning in Problem Solving: An Investigation in the Domain of Dispute Mediation. Ph.D. Thesis. Technical Report #GIT-ICS-85/18. School of Information and Computer Science. Georgia Institute of Technology, Atlanta, GA.

### **Conference Participation and Other Presentations:**

- Memory & Problem Solver Interactions in Natural Problem Solving, invited talk at ARI contractor's meeting, Atlanta, GA, November, 1985.
- Representation of Experience in Long Term Memory, at invited conference on Text Comprehension & Composition: The Role of Mental Representation of Meaning, Center for Research in Human Learning, U. of Minnesota, October, 1985.
- Natural Problem Solving, invited talk at ONR contractor's meeting, Atlanta, Georgia, January, 1985.
- 'If you want my advice...': Some protocols of a memory-intensive task, invited talk at Second Annual Workshop on Theoretical Issues in Conceptual Information Processing, New Haven, CT, May, 1985.

Invited conference participant, Third International Machine Learning Workshop, Skytop, PA, June 24-26, 1985.

A Process Model of Case-Based Reasoning in Problem Solving, presented at the Seventh International Joint Conference on Artificial Intelligence, Los Angeles, CA, August, 1985.

Experience's Roles in Natural Problem Solving. Invited talk given at Louisiana State University, November, 1984; Rutgers University, November, 1984; University of California, Irvine, December, 1984; and UCLA, December, 1984.

Experience's Roles in Clinical Problem Solving. Invited talk given at Stanford University, December, 1984, and University of Michigan, January 1985.

Using Experience to Solve Problems. Invited talk given to College of Science and Liberal Studies, Georgia Institute of Technology, June, 1985; IEEE Computer Society, Atlanta Chapter, November, 1985.

Attached are two copies each of each available paper listed above.

Sincerely,

Janet L. Kolodner  
Associate Professor  
Principle Investigator

## Some Little-Known Complexities of Case-Based Inference\*

Janet L. Kolodner  
School of Information and Computer Science  
Georgia Institute of Technology  
Atlanta, Georgia 30332

### 1. Background

Over the past several years, the research group at Georgia Tech has been engaged in a study of case-based reasoning. In an attempt to be able to come up with results that cut across several different problem solving methods and styles, we have looked at expert task domains, such as psychiatric diagnosis and labor mediation, and common-sense task domains, such as solving everyday resource problems, giving advice about acquiring household appliances, and most recently, meal design.

The most complete implementation of a case-based reasoner to come out of our group to date is the MEDIATOR (Simpson, 1985, Kolodner, et al., 1985), a program that uses case-based reasoning to understand and resolve disputes in a common-sense way. There are several major points illustrated by the MEDIATOR project.

First, it showed that case-based inference is appropriate for any kind of inference that needs to be made, provided the appropriate previous cases are available. The MEDIATOR used case-based reasoning to understand and elaborate problems, to derive planning policies, to derive plans, to figure out what went wrong when feedback signaled that the plan didn't work as expected, and to fix mistakes that had been made.

Second, it showed how at least some reasoning shortcuts are allowed with case-based reasoning. The MEDIATOR has a plan instantiation problem solver, and case-based reasoning allowed specific plans to be chosen without considering more abstract ones.

Third, the MEDIATOR illustrated that if the reasoning process keeps its reasoning goals explicit, then the case-based reasoning process can be directed by those goals. While reminding depends on the description of the whole case, access to parts of the previous case was described as *demand-driven*, where demand is supplied by the reasoner's goals. If, for example, the reasoner was trying to elaborate the goals of disputants, then those portions of the previous case that had to do with disputant goals were accessed and checked for applicability to the new case. When the reasoner was trying to figure out why a particular error occurred, only the explanation of the error from the previous case was accessed from that case. The reasoner's goals, in general, are assumed to be needs to derive values to fill in particular gaps in the representation of the problem description, solution, or evaluation.

There are many problems that the MEDIATOR did not address, however, and in our analysis of the MEDIATOR's capabilities, we are finding out additional things about case-based reasoning. The particular topics we have been concentrating on recently are avoiding mistakes through case-based reasoning, the role of analogy in case-based reasoning, and keeping track of the justifications and conditions for each reasoning decision that is made.\* We are addressing these problems in the context of our JULIA project (Cullingford & Kolodner, 1986), an attempt to design an automated colleague that acts as a caterer's assistant. In this paper, we present our current analysis of the processes involved in case-based inference, illustrating our points with cases from the MEDIATOR.

---

\* This research is supported in part by NSF under Grant No. IST-8317711 and ARO under Contract No. DAAG-29-85-K-0023.

\* Carbonell (1985) calls these *derivations*.

## 2. Case-Based Inference

Making an inference based on a previous case involves recalling a relevant case from memory, determining which parts of that case might be appropriate to make the necessary problem solving decision for the new case, checking the applicability of those portions of the old case to the new case, and, if applicable, making a hypothesis based on the old case, checking its applicability to the new case, and changing the representation of the new case appropriately. Case-based reasoning thus includes the following steps:

1. Locate and retrieve a potentially applicable case from long term memory.
2. Determine which portions of the old case might be applicable to the current one (based on problem solver goals).
3. Derive the targeted value for the current case,\* and propose it. This requires the following considerations:
  - (a) The role of the targeted portion of the previous case in the success or failure of the previously derived plan.
  - (b) Choice between previously-used inference method and previously-used value.
  - (c) Potential applicability of the previously-used inference method or value to the current case.
4. Check the derived value for consistency with the current case.
5. Update the representation of the current case.

In our discussion below, we will concentrate on Step 3, deriving a value for the new case based on the old one. We will assume an appropriate previous case has been chosen (Step 1) and that the problem solver's goals are known, thus directing the reasoner to focus on a particular part of the previous problem to compute a corresponding value for the new one. As is listed above, deriving a value for the new case requires several considerations. The role of the targeted portion of the previous case in the success or failure of the previous case must be checked to determine if there is potential for failure, and if so, what to do about it. In addition, there are several things that can be transferred from a previous case to a new one: the particular value used previously or the inference method that chose that value. Finally, the potential applicability of the previously-used inference method or value must be checked with respect to the current case.

### 2.1. Avoiding Failure

The first step in deriving the value of a targeted feature in the new case is to make sure that the selected portion of the previous case can be used to make such a prediction. While the selection process from Step 1 is responsible for ruling out inapplicable cases, this step is responsible for investigating the applicability of portions of cases. This is done by analyzing the role of the previous case's targeted portion in its success or failure. If the previous case was a success because of the targeted portion of the case, or it is known that the previous case was a success but not why, then the targeted portion of the previous case is judged appropriate to use in deriving the targeted value in the new case. If, however, the previous case resulted in failure, additional analysis is needed before deciding if a suggestion from that case is appropriate. If the targeted portion of the previous case was not responsible for the failure and did not change as a result of reanalyzing the case when failure was discovered, it is probably alright to proceed. If, however, the targeted portion of the previous case was responsible for the failure or changed as a result of reevaluating the case, further analysis is needed, and the value or inference rule targeted from the previous case is often ruled out.

Consider, for example, a problem solver attempting to resolve a dispute over possession of an avocado. Two people both want it. Suppose that the problem solver is attempting to fill in the underlying goals of the disputants and is reminded of the candy dispute. Two kids wanted the same candy bar and the reasoner divided it equally between them by having one divide it and the other choose his half first. Since this worked successfully, the problem solver can make the case-based inference that these people also want to eat their food. Suppose, now, that the problem solver is reminded of another dispute, the orange dispute. In its first analysis of the orange dispute, the reasoner incorrectly used a *default-use*

---

\* In previous papers about the MEDIATOR (Simpson, 1985, Kolodner, et al., 1985), this step was presented as a transfer step. Some value was transferred from the old case to the new one. In our current analysis of case-based reasoning, there is often no transfer. Rather, a hypothesis is computed on the basis of the inference used to compute a value in the old case, and if it is applicable, it is added to the new case.

Inference to infer that both disputants wanted to eat the orange. It turned out that the goal of one of the disputants was to use the peel of the orange to bake a cake. The *default-use* inference applied to the orange as a whole led to the solution to that dispute failing, and in the second analysis, it was applied correctly to two *parts* of the orange: the fruit and the peel. Based on this reminding, the reasoner is alerted that if the avocado has several parts, the goals of the disputants may have something to do with the parts and not necessarily with the avocado as a whole. It may be judged in this case that inference based on the parts is inappropriate (since one rarely plants avocado seeds), but the potential for failure is flagged and two alternative solutions are presented.

Evaluation of applicability of values or inference methods from failed cases happens as follows: If the previous case was reanalyzed and a successful solution found, the problem solving episode representing successful solution is also retrieved from memory (there will be a pointer to it from the failed episode).<sup>\*</sup> The inference rules and justifying conditions used to infer the targeted portion of each case are recalled. A check is made in the current case to see if the justifying inference rules and conditions from the failed case hold. The knowledge necessary to determine this may not have been computed yet, so this step may require significant computation. This extra computation, while significant, is done only when a previous case points to the need to look out for a problem. It is a preventative measure to aid in avoiding failure. If this reasoning determines that the conditions of the previous case do hold again, the inference method and value used previously are ruled out. If not, suggestions from that case may still be applicable, and the process above is carried out. Either way, another check is made to see if the problem solving episode representing successful resolution of the previous problem can pass the selection test in Step 1 (i.e., are its goals appropriate?). If so, it is selected for case-based reasoning and a value is derived for the targeted portion of the current case using it. If both the failed and the successful versions of the previous case can both make predictions in this step, then another decision must be made concerning which is the better proposal.

Going back to the reminding of the orange dispute, the disputant goals are targeted by the case-based reasoner. The inference used to compute it is recalled. It was a *default-use* inference applied to the orange (the disputed object) as a whole, which is appropriate to the new case (when applied to the avocado). The successful rendition applied *default-use* to the parts of the orange, its fruit and its peel. This also is possible in the case of the avocado, if one considers the fruit and the seed (which can be planted). Either is a possibility, and further reasoning must be done to decide which is better.

## 2.2. Inference Method or Value?

One other selection must be made before a value can be derived for the new case. A choice must be made between the two possible case-based derivations: transfer a value or use a previously-used inference method. Under some circumstances, the *value* used previously is suggested at the end of this step. For example, when the problem solver is attempting to choose a skeletal plan, and when the previous case is similar in plan-related details to the current case, the skeletal plan used previously is suggested by this set of steps. Similarly, when a diagnostic problem solver is trying to diagnose a medical disorder, the same disorder seen in a previous case will be suggested if the disorder-related details of the two cases are the same.

Under other circumstances, the *inference method* used previously is suggested, and the targeted value is derived for the new case using that inference method. Consider a problem solver attempting to solve the orange dispute. It is trying to determine why they want it, and is reminded of a dispute over a candy bar. It could transfer the goals of the people with respect to the orange from the candy bar dispute: they want to eat it. This seems like a silly way to make this inference, however, since this inference can be made easily without using the previous case.

<sup>\*</sup> In the representation we are currently using, each full analysis of a problem is kept separately with pointers between them. Thus, the representation for the orange dispute is actually represented as two cases. The first is the one that failed, where one set of assumptions was made about the goals of the disputants. That one includes the mistaken problem description, the suggested plan (cut it in half), feedback after suggesting the orange be cut in half, and the analysis of what went wrong. It tags the failure as a *wrong-goal-inference*, in essence saying that the original inference about the goals of the disputants was responsible for the failure. It points to a second problem solving episode in which the problem is described as one where the disputants have the second set of goals, and the plan that goes with that is recorded there. Reminding may be of either the successful or the failed case. When reminding is of the successful case, the reasoning that allowed a successful solution to finally be found is bypassed and a good solution is suggested immediately. The problem solver is never alerted of possible problems. When reminding is of the failed case, the problem solver is alerted and the analysis described is done.

Transferring the inference method instead, however, makes case-based inference more sensible. In the candy-bar dispute, a *default use* inference was used to infer that the kids were going to eat the candy bar. The candy-bar dispute suggests that this same inference rule should be used for the orange dispute. In doing this, it confirms that the default inference is the appropriate one, and that there are no special circumstances it knows about that would rule it out or lead to a different suggestion. A similar case-based inference can be made, based on the same candy dispute, when encountering a dispute over possession of any other kind of object that can be split and still used effectively: the inference would be that the object was to be used by the disputants in its default way. While these examples show a previous case suggesting and thus confirming use of the default inference, in other cases novel inferences suggested by a previous case result in inference of something other than what a from-scratch reasoner would attempt first. Use of an inference rule used in a previous case generally results in an *analog* to the value used previously being suggested, with the inference rule providing part of the mapping. (e.g., food, in the first case, is eaten, its default use; land, in the second case, is settled, its default use).

The heuristic we propose for deciding whether the inference method or the value should be taken from the previous case is the following:

1. If the inference method used previously is available, and if its justifications hold in the current case, use the inference method used previously to derive a value for the current case.
2. If the inference method is unavailable or if there is not enough knowledge to judge whether or not it can be appropriately applied to the current case, see if the value from the previous case is appropriate to the current one. If so, propose it.
3. If the value is inappropriate, and there is no known reason to rule out the inference method, then try applying it to the current case.

In essence, this heuristic says to try the inference method if there is a way to determine that it will probably work without doing its computation. Otherwise, check the value for appropriateness, an easy job, and if it is inappropriate, then go through the work of using the inference method, if nothing is known that rules it out.\*

### 2.3. Justifying Use of Previous Inference Methods and Values

Deciding whether to use the previous value or the previous inference method requires investigation of how the previous value was derived, if it is known, or conditions justifying its use. Only if the problem solver knows how the previous value was derived can the inference set used to derive it be used for the new case. Otherwise, the best that can be done is to check if the previous value is appropriate to the new case.

To describe how an inference method can be justified for use, we must define what an inference method is. An inference method may be a simple inference rule with a set of antecedent clauses and a set of consequent clauses. In this case, it is justified if its antecedent clauses hold in the new case. One inference rule used in the MEDIATOR, for example, is the *infer-goal-from-default-function* rule (we have called it *default-use* up to now). This rule is applicable (its antecedent) when an object is possessed or controlled by a person, or when a person wants to gain possession or control of an object, and when the person's goal with respect to the object is required. It is executed (its consequent) by retrieving the default function from the *function* slot of the object and then inferring that action as the goal of the person. It can be used to infer that a piece of fruit will be eaten, that a book will be read, etc.

An inference method may also be a chain of inferences, in which case justifying use of the inference chain may require justifying use of each of the inferences in the chain. In some cases, this requires doing the entire computation for the new case. Carbonell's (1985) derivational analogy addresses this problem, and we do not concentrate on multi-step inference chains here.\*

\* Since the MEDIATOR transfers only values, this heuristic has not been tried out in the MEDIATOR. We have tested it with several of the MEDIATOR's examples, however, as well as some other examples, and we are currently testing it in the JULIA program.

\* Because the MEDIATOR is a hierarchical problem solver whose reasoning goals are defined by the problem solver itself and not the problem, its inference chains for any particular goal it had to achieve were fairly shallow. Our other problem solvers also have fairly shallow chains of inference when we divide the problem into parts. We thus have not considered this problem. Carbonell needed to address this problem because his systems had little hierarchical control on their problem solving.

Another source of justifications is the set of conditions under which a concept or inference was chosen. In general, consideration of a previous value or inference rule is justified when the conditions under which it was chosen are also in force currently. These conditions include preconditions or recognition criteria of a targeted value, and other problem solving goals, policies, or constraints served by the value. A particular inference rule might have been a good choice in a previous case, not only because it was applicable (i.e., its antecedents held) but also because some other inference rule had already predicted something consistent with it about the value it derived.

A value or inference method is worth consideration if its conditions of applicability hold (e.g., preconditions for a plan, antecedent for an inference rule, recognition criteria for a classification). Its use is further justified if special conditions under which it was chosen previously are in force in the new case. If previous special conditions are not in force, it may still be appropriate but not best. Since this step generates hypotheses, it may be a reasonable hypothesis that gets ruled out later.

Use of a previously-used inference method requires that the problem solver explicitly store justifications and inference methods employed to derive each value it computes. There are several ways to do this. The MEDIATOR keeps the whole set of inferences used in solving a problem on its blackboard and attaches the reasoning chain with its case before storing the case in memory. In our JULIA program, we keep track, for each value present in a representation, the inference rule used to derive it (if one was used) along with its bindings and the sources of those bindings (e.g., the slots they came from), the set of choices available when the value was chosen, the set of values that had been previously ruled out, and any conditions (goals) the problem solver was trying to achieve that this value contributed to.\*

The *value frame*, as we are calling it in JULIA, for the goal of sister1 in the orange dispute (failure version) would look something like the following:

```
value: sister1 <=> "INGEST" <-o- fruit-of (orange1)
suggested values: same as value
ruled-out values: nil
justifying-inference:
  inference-rule: default-use
  binding: orange1
  binding-source: disputed-object slot of problem specification
conditions: Goal (sister1) is to possess orange1
```

The *value frame* for her goal in the orange dispute (successful version) would look something like the following:

```
value: sister1 <=> $BAKE using peel-of (orange1)
suggested values: same as value
ruled-out values: sister1 <=> "INGEST <-o- fruit-of (orange1)
                  conditions: sister1 says she wants the peel to bake
justifying-inference:
  inference-rule: special-use
  binding: peel-of (orange1)
  binding-source: part-of object filling disputed object slot of ...
conditions: Goal (sister1) is to possess orange1
           sister1 says she wants the peel to bake
```

As an aside, this representation, which keeps track of suggested and ruled out values along with inference rules and conditions that support chosen values, gives a way to record alternatives when several are suggested. The justifying inference rule will then be an inference that chooses between the possibilities on the basis of some set of criteria.

---

\* Carbonell's (1985) derivational analogy keeps track of similar information. Another way to do this is by using the standard bookkeeping provided by a truth maintenance system.

### **3. Some Conclusions**

#### **3.1. Reasoning Shortcuts**

Previous cases provide inference methods or values to use in reasoning about a new case. Either way, there is potential to save a great deal of computation. If inference methods are provided, the inference method doesn't need to be derived. If values are provided, the from-scratch inferencing doesn't have to be done. This all depends, of course, on getting appropriate reminders and not too many of them so that the computations involved in case-based reasoning do not become overwhelming.

#### **3.2. Avoiding Mistakes**

When a previous case in which an error was made is recalled, it flags the potential for a similar mistake. It also suggests the correct solution finally found in the previous case. The combination of these helps the problem solver to avoid repeating mistakes and suggests shortcuts in reasoning that avoid the trial and error of previous cases. Being able to avoid mistakes in this way requires that cases be indexed by descriptive features that are likely to be encountered and recognized in later cases.

#### **3.3. Focus**

When a previous case flags an error, it also directs reasoning to that part of the problem that was responsible for the previous error. If inferences being considered require knowledge that is not yet available, reasoning is also directed to fill in that knowledge.

#### **3.4. Transfer or Analogy?**

All of our previous work on case-based reasoning, and all of Carbonell's work talks about *transfer* as being the primary case-based reasoning process. A previous value or schema or plan step was borrowed or transferred from the previous case to the new one, perhaps being reinstated along the way. Even Carbonell's (1985) derivational analogy discusses under what conditions it is appropriate to transfer a plan step from a previous plan. Our analysis of case-based reasoning processes, however, shows that very often the previous case provides an inference method rather than a value. The inference rule, when applied to the new case, bound as specified in the old case, derives an analog to the old value for the new case.

We started thinking about this problem as a result of comments by Jerry DeJong at the Allerton Workshop on Similarity and Analogy in which he pointed out that much of the work people are calling analogy, including analogical problem solving, is only reinstatement and is not analogy at all. It turns out both are necessary, and which is done depends at least partially on how close the two cases are to each other and how much justification is available for the old value.

#### **3.5. Which Comes First -- Case-Based or From-Scratch Inferences?**

We can now conclude that case-based inferences should be attempted before from-scratch inferences if a relevant case is available. At worst, the case-based inference provides justification for use of the default inference. In better cases, it points out the potential for failure or suggests a novel way to solve the problem.

#### **3.6. Major Bottlenecks**

Case-based reasoning is a lot harder than just taking a previous value and transferring it to a new case. Many of the computations are significant. For this reason, there must be a way to limit the cases the case-based reasoner sees. While we've done studies of how to get cases into and out of memory, a big problem we continue to see is that the memory returns too much. Something has to be done to solve this problem. A more major bottleneck, however, is the sheer complexity of memory processes. In all of our experiences with case-based reasoners, it is the memory that slows down the process. Any memory with full traversal and update apparatus is horribly slow, at least when implemented on serial machines. It is probably necessary to show that a memory for cases can be made fast before case-based reasoning will be taken very seriously as a viable reasoning method for automated systems (Kolodner & Cullingford, 1986).

#### 4. References

- Carbonell, J. G. (1985). Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition. In Michalski, R. S., Carbonell, J. G., & Mitchell, T. M., *Machine Learning II*, Morgan Kaufmann Publishers, Inc.
- Cullingford, R. E. & Kolodner, J. L. (1986). Interactive Advice Giving. In *Proceedings of the 1986 IEEE International Conference on Systems, Man, and Cybernetics*.
- Kolodner, J. L. & Cullingford, R. E. (1986). Towards a Memory Architecture that Supports Reminding. In *Proceedings of the 1986 Conference of the Cognitive Science Society*.
- Kolodner, J. L., Simpson, R. L., & Sycara, K. (1985). A Process Model of Case-Based Reasoning in Problem Solving. In *Proceedings of IJCAI-85*.
- Simpson, R. L. (1985). A Computer Model of Case-Based Reasoning in Problem Solving: An Investigation in the Domain of Dispute Mediation. Ph.D. Thesis. Technical Report No. GIT-ICS-85/18. School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA.

020114

<b>NATIONAL SCIENCE FOUNDATION</b> Washington, D.C. 20550		<b>FINAL PROJECT REPORT</b> NSF FORM 98A			
<i>PLEASE READ INSTRUCTIONS ON REVERSE BEFORE COMPLETING</i>					
<b>PART I-PROJECT IDENTIFICATION INFORMATION</b>					
1. Institution and Address Georgia Institute of Technology School of Info. & Comp. Science Atlanta, GA 30332-0280	2. NSF Program Information Science and Technology	3. NSF Award Number IST-8317711	4. Award Period From 6/15/84 To 11/30/87	5. Cumulative Award Amount \$231,000	
6. Project Title Extracting Information from Experience-Driven Learning					
<b>PART II-SUMMARY OF COMPLETED PROJECT (FOR PUBLIC USE)</b>					
<p>This report is in three parts: Part one summarizes work on case-based reasoning. Part two summarizes our approach to representing procedural knowledge declaratively. Part three lists papers published as a result of research done on this grant.</p>					
<b>PART III-TECHNICAL INFORMATION (FOR PROGRAM MANAGEMENT USES)</b>					
1. ITEM (Check appropriate blocks)	NONE	ATTACHED	PREVIOUSLY FURNISHED	TO BE FURNISHED SEPARATELY TO PROGRAM	
				Check (✓)	Approx. Date
a. Abstracts of Theses		x (1)			
b. Publication Citations			* 18		
c. Data on Scientific Collaborators	X				
d. Information on Inventions	X				
e. Technical Description of Project and Results					
f. Other (specify)					
2. Principal Investigator/Project Director Name (Typed)  Janet L. Kolodner		3. Principal Investigator/Project Director Signature		4. Date	

\*Sent under separate cover to Dr. Joe Deken of NSF (2 copies each)

LIBRARY

## Extending Problem Solver Capabilities Through Case-Based Inference

JANET L. KOLODNER

(KOLODNER@GATECH.EDU)

School of Information and Computer Science, Georgia Institute of Technology,  
Atlanta, GA 30332 U.S.A.

### Abstract

In case-based reasoning, the problem solver makes its inferences based directly on previous cases rather than by the more traditional approach of using general knowledge. Case-based reasoning results in several enhancements to problem solving behavior over time. First, recall of previous failures warns the problem solver of potential for failure and allows the problem solver to avoid making mistakes made previously. Second, previous decisions that have been made previously are suggested to the problem solver so that its decisions do not have to all be made from scratch. This lessens the search space and also is a way of shortcutting the constraint satisfaction process. Third, if abstract schemata can be derived from cases that have been seen previously, generalized knowledge can be augmented. This allows real shortcuts in problem solving. Decisions that previously took several reasoning steps to make may be possible through application of a generalized schema.

### 1. Introduction

Much of the problem solving people do from day to day involves consideration of previous similar situations. Access to previous experiences helps the problem solver anticipate and avoid repeating mistakes and aids in the derivation of reasoning shortcuts. The work reported in this paper is an investigation of the processes involved in making inferences based on individual past experiences. The set of processes that are employed to do this are called *case-based reasoning processes*. Our goals in this endeavor are manifold. First, we wish to understand the processes involved in doing this type of problem solving analogy. Apparently, it comes easily and naturally to people. But it is not a process that is normally employed in problem solving or learning systems. Our second goal parallels the first. As a result of learning about the processes involved in case-based reasoning, we want to be able to specify what an automatic reasoning system needs to employ these processes and under what circumstances they ought to be applied.

To illustrate case-based reasoning, we give two examples from the meal planning domain. Case-based reasoning, in these examples, is in the context of problem solving. While attempting to solve the problem of deriving an appropriate meal for a client, the caterer, who is the problem solver, is reminded of several previous meals and uses those to aid the problem solving, first to anticipate and avoid a problem she had previously when she did not know to plan for vegetarians attending a dinner party. In this case, case-based reasoning directs her to find out the information that she had been missing the last time that had caused a problem in the previous meal. In the second instance of case-based reasoning, the previous case is used to make suggestions about cuisine and what to serve at the meal. Taking the constraints set up by the current problem into account, the caterer is reminded of a meal in which some of the same constraints were active and begins planning the new meal based on that. Taking into account the differences between the two situations, the caterer comes up with a plan appropriate to the current meal. Without the previous cases, the problem solver would have repeated its previous mistake of not planning for vegetarians when it was possible some would be at the party, and it would have had to derive a meal for the party from scratch. Previous cases thus give the caterer a chance to improve her performance in two ways.

Client: I'm having a party for my research group next Saturday. There will be about 20 people.

...

Caterer: (*Remembering another party with graduate students, where there was no adequate provision for the one vegetarian in the group*) Are there any vegetarians in the group?

Client: Yes, ...

...

*Caterer: Last time you had a buffet dinner and were on a limited budget, you served a combination of Indian and Chinese dishes. We could do that again, but substitute several veggie dishes for some of the meat ones.*

Over the past several years, we have investigated case-based reasoning in a variety of domains, some common-sense and some expert. Our most recent endeavors are in the domains of labor mediation (Sycara, 1985), meal planning (Cullingford & Kolodner, 1986; Kolodner, in press), and planning for acquisition of household products (Turner, 1986). Our illustrations in this paper will come from the meal planning domain. Our program, called JULIA (Cullingford & Kolodner, 1986; Kolodner, in press) employs a combination of case-based reasoning, constraint propagation, and problem reduction problem solving to derive menus for meals. It uses its reminders of past failures to warn it of the potential for error and to suggest means of avoiding errors made in the past. It uses its reminders of past successes to derive suggestions as it is planning a meal. Previous cases can suggest whole meals, parts of meals, or attributes of meals, depending on the problem solver's focus at the time reminding happens.

## **2. Making a Case-Based Inference**

Making a case-based inference, in the simplest case, includes the following set of steps:

1. Recall a previous case
2. Focus on appropriate parts of that case
3. Use those parts of the previous case to derive an appropriate decision for the new case

### **2.1. Recalling a previous case**

Recall of a previous case is done by probing the memory. According to Schank's (1982) MOPs theory, understanding of a new input (case) includes finding the best knowledge in memory that can be used to make predictions from it. Finding this knowledge is equivalent to integrating the new case with what is already in the memory. As reasoning is going on, according to this theory, memory is constantly being probed and updated, the case is getting better integrated, and better knowledge to use in making predictions about the case is being derived.

According to the same theory, generalized knowledge and individual cases are organized together in the same memory (see Kolodner, 1984; Lebowitz, 1983 for means of implementing such a memory). As a result, as a case is being understood and integrated into memory, both generalized knowledge and individual cases become available to use in further processing it. The case is integrated to the most specific possible place in memory. It may closely fit a generalized category or may be most similar to a specific previous case. When it closely fits a generalized category, the generalized knowledge associated with that category is used for problem solving. When it most closely fits a case, that case becomes available for case-based reasoning. Because the current case can be better integrated into memory as more is derived about it, new cases not previously considered are encountered in memory and become available for case-based reasoning as problem solving progresses. Usually, cases encountered later in problem solving are similar to the current case in more specific ways than the ones recalled earlier in problem solving. The problem solver thus may use several cases in coming up with a problem solution, each related more specifically to the current case than the one that came before.

In JULIA, recall is realized by a memory process running concurrently with the problem solver. A global knowledge structure (called a blackboard) holds the representation of the current problem situation, including the problem statement and the evolving solution. The memory prober uses the problem representation to derive its cues for searching memory. As the problem statement is filled out in more detail, the memory prober has additional cues available to it, and as a result new cases may become available for case-based reasoning as problem solving progresses. Because several cases may be available at once, JULIA needs a way to choose from among the available cases. It focusses first on those that share the most goals with the current case. To choose the best of those, it chooses those that share the most and most important constraints. Other features are taken into account only after that.

## 2.2. Focussing on appropriate parts of the recalled case

Any particular case that is recalled could be quite large. The entire case is not necessary for making a case-based inference. In fact, the whole case with all of its details is too cumbersome to work with. Rather, the parts of the case that have relevance to the new case are the ones to focus on. There are two ways this can be done. If the previous case was successfully resolved, the problem solver's current reasoning goals determine focus. If the previous case resulted in failure, focus is on those parts of the previous case that were responsible for the failure.

Let us examine the case where the previous problem was successfully resolved first. In this case, it is the problem solver's current reasoning goals that determine which parts of the previous case to focus on. In general, problem solving involves achieving a number of goals that are often reduced to subgoals. A reasoner that has to plan a meal will have subgoals associated with finding a cuisine, finding an appetizer, finding a main course, finding a dessert, etc. Each of these subgoals may itself be reduced to several subgoals. Finding an appetizer, for example, may require coming up with a main ingredient, coming up with a preferred preparation method or flavor enhancer, and then finding a recipe. Given the current goal of the reasoner, focus is directed to those parts of the previous case that are relevant to fulfilling that goal. Thus, when a caterer is reminded of a case while trying to determine cuisine, it will focus on the cuisine of the previous meal and the reasons it was a good choice in that case to see if the cuisine chosen previously is appropriate for the current case. When it is trying to come up with a dessert, it will focus on the dessert served in a previous case it is reminded of and the reasons that dessert was a good choice.

Where do these reasoning goals come from? One can think of a general purpose reasoner that is at least initially in charge of reasoning goals. As a problem is being reasoned about, the goals and subgoals that must be achieved to resolve it are derived by that reasoner. In Carbonell (1983, 1988), that reasoner is a means-ends analysis problem solver and therefore derives its subgoals by comparing the current and goal states, deriving their differences, and setting up subgoals of reducing those differences. After a case is recalled, the progression of subgoals that were used in solving that case are used to complete processing of the current case. Another method, used by Hammond (1986), Kolodner (1985), Kolodner, Simpson, & Sycara (1985), and Simpson (1985), is to set the sequence of goals a priori. The problem solver progresses through the sequence of goals, attempting to achieve each one. If a case is available when a goal is encountered, that case is used to achieve the goal, otherwise from-scratch methods are used. After one goal is achieved, the next in the set sequence is attempted.

JULIA is implemented yet another way. The case-based reasoner runs in conjunction with a constraint satisfier, problem reduction problem solver, constraint propagator, and conversational controller. A goal scheduler keeps track of the current goals of the system. Goals are posted on the global blackboard for all processes to see, and any that have the potential to achieve a posted goal attempt it. Any time one of those processes needs information, it posts its goals with the goal scheduler. This means that the case-based reasoner can take its direction from any of the processes requiring inferences to be made.

The goal scheduler initially takes its direction from a problem reduction problem solver. The problem reduction problem solver knows, for example, that to plan a meal, one must first derive meal constraints and descriptors, then plan the main course, then plan the salad, appetizer, and dessert. When memory returns a case, the case-based reasoner attempts to achieve the system's current most specific goals by using the case. When that goal is achieved, it may use the same case to attempt to achieve other active goals. The case-based reasoner itself might also need to achieve a set of subgoals that cannot immediately be achieved by the current case. When this happens, it posts those goals with the goal scheduler.

Suppose, for example, that the current most specific posted goals are "choose main dish", "choose side dish (veg)", and "choose side dish (starch)", all subordinate to "choose main course". If the memory returns a case at this point, the case-based reasoner will first focus on the main dish of the previous course to see if anything can be adopted from it. If, after suggesting the main dish used previously, no new cases are returned from memory, it will use the same case to attempt to achieve its side dish (veg) goal. In general, case-based inference can be used to achieve goals at any level of detail or abstraction, and when this happens, full problem reduction is shortcuted.

When the previous case resulted in failure, focus is directed to those parts of the previous case that were responsible for that failure. In essence, when a failed case is recalled, the current goals of the problem solver are put on hold and focus is on avoiding the error made previously. In the dialog above, for example, the caterer was trying to fill in meal descriptors (constraints) before beginning the job of choosing dishes for the meal. Meal descriptors include finding out cost, deriving

cuisine, finding out how serving might be done, and finding out how many people will be in attendance. In the course of attempting to achieve these subgoals, she was reminded of a previous failed case in which vegetarians were not accounted for in the planning. The case-based reasoner focuses on that failure and attempts to find out if it is possible for that failure to happen in the current case, and, if so, attempts to prevent it. We explain the processing necessary for this task below. After dealing with a failed case, focus may revert back to the goals that were active before finding the failure, or the problem statement might be sufficiently changed by the process of preventing the failure that problem solving goals must be rederived using the new problem statement.

### 2.3. Making the case-based inference

At this point, we have an old case and a problem solver goal, we have focused on a part of the old case that is to be used in achieving that goal, and we have the case we are currently working on. When the previous case was successful, the purpose of the case-based inference is to achieve the goal for the new case based on the old one. For example, if the current goal is to choose a cuisine, and a previous case is available with similar meal constraints and meal description, the case-based reasoner examines the choice of cuisine from the old case (the way the cuisine goal was achieved in the previous case) to derive a way of achieving that goal in the new case. When the previous case resulted in failure or failed expectations, the purpose of case-based inference is to judge whether the same error is possible in the current case and to avoid repeating it.

Let us consider first the processes of case-based inference when the previous case was successfully resolved. There are several processes available for making such case-based inferences:

1. Transfer the solution that achieved the goal in the previous case.
2. Transfer the solution that achieved the goal and modify it based on differences between the current and previous case.
3. Transfer the inference method by which the previous goal was achieved.
4. Create an abstraction of the problem descriptions from the two cases, extend it to fit the solution to the previous case, and apply it to the new case to create a solution.

The process to be used depends on a number of considerations: Is there a value that, when derived, will achieve the goal, and if so, is that value available in the old case? Do we know how that value was derived for the old case? Was it by an "easy" or a "complex" set of reasoning steps? Do we know why the value from the previous case was appropriate? Do we know why the method of deriving that value previously was appropriate? If achievement of the goal is not done by simple derivation of a value, do we have a generalized schema that explains how the goal was achieved previously? If no schema, do we have the set of steps? Is our goal to derive a plan or is it to derive a feature value? Is the problem solution easily decomposable into non-overlapping parts? Or is the previous solution one that integrates the achievement of several goals simultaneously?

When the goal to be achieved can be achieved by choosing a single value or fully-instantiated frame, Method (1) is the one of choice. This method is the simplest, and is the one employed by Carbonell (1983), Hammond (1986), Kolodner, et al. (1985), and Simpson (1985). It requires a lookup of the solution to the active goal in the previous case and then consideration of whether that solution will work in the current case. Suppose, for example, that JULIA has the goal of choosing a main dish. It is appropriate, in this case, to recall the main dish used previously and consider whether it is appropriate for the current case. This method is also appropriate when the inferences necessary for achieving the goal were quite complex and only the solution itself is necessary as a hypothesis. This might happen in meal planning if there was a lot of trial and error reasoning that went into choosing a particular dish for a previous meal. Only the dish need be considered. The long reasoning chain followed previously does not have to be repeated.\*\*

When the solution transferred from a previous case does not fit the new case, Methods (2) or (3) are necessary. We consider Method (3) first and consider Method (2) later since it is used in other cases also. According to Method (3), the chain of inference used to derive a solution to the current goal in the previous case is repeated in the new case. When the

---

\*\* This is also the case in medical diagnosis. A previous diagnosis acts as a hypothesis that must be confirmed, and at least initially, direct transfer of the solution (diagnosis) from the previous case is all that is necessary.

goal is a high-level one and the task is planning, Method (3) is equivalent to Carbonell's (1986) derivational analogy. Let us consider, however, the case where the goal is a fairly specific one: "choose main dish (vegetarian)". One way this can be done is to choose a dish that normally has meat but to use the vegetarian version of it.\* If, when the problem solver is trying to achieve this goal, it is reminded of a meal in which vegetarian lasagne was chosen as the main dish in this way, it might attempt the same inference method: choose a dish central to the chosen cuisine, then search the set of vegetarian recipes for a version of it that does not have meat. In essence, using this method, the conditions under which a previous decision was made are taken into account, and the case-based inference tends to be a transfer of the method of decision making or the inference rules used previously rather than the solution. Because this method is more time-consuming than the transfer method (1), JULIA uses it only if transfer has been ruled out or if solutions require different features.

When the previous solution is not easily decomposable into non-overlapping parts, when it integrates the achievement of several goals simultaneously, when a high level goal with a routine achievement method provides the current focus, or when a high level goal that cannot easily be decomposed provides the current focus, Method (2), sometimes called *comparison-based reasoning*,\*\* is the one of choice. Using this case-based reasoning method, the solution that achieved the goal previously is transferred from the previous case and modified based on differences between the current and previous case. While this method has not been implemented in JULIA, we have found it useful for the task of labor mediation (Sycara, 1985), in which the previous solution integrates partial fulfillment of many simultaneous competing and conflicting goals and cannot be easily decomposed into parts. We (Turner, 1986) and others (Alterman, in press) have also found it useful in achieving goals with a routine method of achievement when a case the specializes that routine is available.

Method (4) is more schema-based (see, e.g., Holyoak (1984)). In this method, the current and previous cases are compared and a schema describing the similarities of the problem statements is described.\*\*\* The schema must be such that it can be used to describe both problem statements. The schema is then broadened to describe the solution to the previous problem, and the new problem is solved by applying the schema to that problem. In principle, it should be possible through this method to derive real problem solver shortcuts by storing the derivations of the reasoning steps in the schema where they do not have to be considered during later problem solving except when something goes wrong. This is not possible with any of the other methods by themselves.

While these four methods are the ones that are applicable when the previous case resulted in success, additional reasoning must go on when the previous case resulted in failure (Carbonell, 1986, Kolodner, in press). In this case, the conditions under which previous values were computed and the set of steps used to make decisions are checked against the new case to see if the same potential for failure exists. The previous case may also provide suggestions to the problem solver of how to proceed. In essence, the reasoning that goes on here is a special case of derivational analogy.

In short, the steps that must be followed to capitalize on a previous failure are\*: (1) determine what was responsible for the previous failure, if possible (this may already be recorded, and if not, some short amount of time is spent attempting to derive it), (2) direct reasoning focus to the decision in the new problem that is analogous to the one that caused the failure in the previous one (this may be the one currently being focussed on or one that its correct solution is dependent on), (3) check for the potential for the same failure in the new case, either by seeing if the explanation of the previous failure holds in the new case or by checking the reasons why the previous decision was made and seeing if the same justifications might apply in the new case (this step may require additional information gathering), (4) if not, potential for error is not there, so return to the interrupted step and keep going, (5) if so, rule out the previous errorful decision as a possibility for the current case, and if the previous case was finally resolved correctly, determine if the decision made when it was resolved correctly is applicable to the new case, (6) if so, use it as a suggestion for a case-based inference, (7) if step 2 redirected focus, then redo whatever decisions must be redone as a result (i.e., follow dependencies) and return to the reasoning step that was interrupted.

\* We are not considering the ins and outs of the planning necessary to do this. See Hammond (1986) for a description of that. Considered Hammond's way, this goal would be a fairly complex one. We are assuming that the system is not deriving the recipe, but rather is choosing from among recipes it has available.

\*\* Term due to Gary Klein.

\*\*\* Work being done by Hong Shinn.

\*Of course, it is more complex than the set of steps shown here, but these steps form the core of the processing. See Kolodner (in press) for more detail.

This is the procedure used by the caterer in the example in Section 1 to determine the need to ask if any guests were vegetarians. After the errorful situation was encountered, the first step was to determine what was responsible for it. In this case, the problem was that the vegetarian constraint was missing from the problem description when problem solving was going on. Focus is directed to constraints on food selection. Checking for the potential for failure in the new case, we find that we do not have any food constraints listed. Since the previous case failed because of lack of knowledge and we are missing the same knowledge in this case, there is a potential for error, and the information necessary to avoid the error is gathered by asking if there are any vegetarians.

#### 2.4. Representational support

Because this processing requires knowing why previous decisions were made, what other decisions previous decisions were dependent on, and what was responsible for previous failures, there must be both a representational system and a bookkeeping system that keep track of this knowledge. Our solution to the representational problem is to have "value frames" (Kolodner, 1986) associated with each value recorded by the system\*. Each time the problem solver makes a decision, it records its decision in the value slot of the value frame and also records what led it to that decision. Value frames include facets for a value, other values that were suggested as alternatives, ruled out values, conditions that were considered in choosing the value, and the inference rule or method or set of steps used to make the decision. Each inference rule that is recorded has three parts to it: the rule body, the bindings that were used in this instance, and the source of those bindings (i.e., where in the problem description can the values used in the bindings be found). The knowledge kept in value frames supports both derivation-type case-based inference (i.e., Method (3)) and avoidance of previous mistakes.

While value frames keep the justifications for each decision, pointers in the other direction are needed when the problem solver needs to retract an already-made decision. Our solution has been to integrate a truth-maintenance type system with our problem solver (Kolodner, in press).

### 3. Implementations

The earliest implementation of a case-based reasoner to come out of our group is the MEDIATOR (Simpson, 1985, Kolodner, et al., 1985), a program that uses case-based reasoning to understand and resolve disputes in a common-sense way. There are several major points illustrated by the MEDIATOR project. First, it showed that case-based inference is appropriate for any kind of inference that needs to be made, provided the appropriate previous cases are available. Second, it showed how at least some reasoning shortcuts are allowed with case-based reasoning. Third, the MEDIATOR illustrated that if the reasoning process keeps its reasoning goals explicit, then the case-based reasoning process can be directed by those goals.

Our most current implementation of the processes described in this paper is JULIA (Cullingford & Kolodner, 1986, Kolodner, in press), an attempt to design an automated colleague that acts as a caterer's assistant. JULIA integrates several processes, as described above. In the following example of JULIA solving a problem, we see some of the processing JULIA does.

The dialog begins with JULIA introducing herself and the user stating her problem.\*

```
<JULIA> Hello. I'm JULIA, a Caterer's Advisor. May I help you?
USER =>
  I'm having a dinner next week for my research group.
  There is no room for all of them in the dining room.
<JULIA>
*** The initial problem is:
**** Frame #<FRAME 34502032> ****
Name: M-MEAL774
```

\*Value frames hold essentially the same things Carbonell (1986) claims are necessary for derivational analogies.  
\* When JULIA is complete, all input will be in natural language. At the moment, JULIA's natural language component is not yet connected, so input to JULIA is a conceptual representation and output is canned phrases. We show the English equivalent to the input here.

Isa: (<M-MEAL>)  
Category: INDIVIDUAL  
Slots:  
ACTUAL-RESULT: NIL  
CHARACTERS: (?HOST ?GUESTS ?PARTICIPANTS)  
CONSTRAINTS: (<C-LIMITED-SPACE778>)  
DEFINED-SLOTS: NIL  
DESCRIPTOR: NIL  
EXPECTED-RESULT: NIL  
FOLLOW-UP: NIL  
GOALS: (<E-EAT778> <S-HUNGER777>)  
GUESTS: < \*JLK\*S-GROUP >  
HOST: < \*JLK\* >  
ORDER: NIL  
PARTICIPANTS: (?HOST ?GUESTS)  
SETTING: < \*JLK\*S-HOUSE >  
STEPS: NIL  
TIME: NIL

JULIA's problem reduction problem solver starts out in charge. It assumes that its goal is to instantiate a meal, and it adds the appropriate task and goal to the task tree and goal chain. It then immediately reduces that task to its component parts and begins working on the first of them, the meal descriptor task. Meal descriptors include the meal's cuisine, the style of presentation, and the placement of the guests when they eat.

\*\*\*\*\* Planning for the task MEAL-TASK775 .....  
TASK-TREE = ((<MEAL-TASK775>))  
The active goal chain is ( M-MEAL )  
Reducing the task MEAL-TASK775 to subtasks (<MEAL-DESCRIPTOR-TASK>  
<MAIN-COURSE-TASK> <APPETIZER-TASK> <SALAD-TASK> <DESSERT-TASK>)

\*\*\*\*\* Planning for the task MEAL-DESCRIPTOR-TASK .....  
TASK-TREE = ((<MEAL-DESCRIPTOR-TASK> <MAIN-COURSE-TASK> <APPETIZER-TASK>  
<SALAD-TASK> <DESSERT-TASK>) (<MEAL-TASK775>))  
The active goal chain is ( MEAL-DESCRIPTOR =< M-MEAL )

JULIA works on some of the meal descriptors and gets to the point where it needs to formulate a cuisine, but it has no information to go on. So it asks.

Constraint Satisfaction .....  
Checking constraints for slot CUISINE of MEAL-DESCRIPTOR783 ...  
Instantiating slot CUISINE of frame MEAL-DESCRIPTOR783 ...  
<JULIA> What cuisine would you like?  
USER => mexican  
<JULIA>

Constraint Satisfaction .....  
Checking constraints for slot CUISINE of MEAL-DESCRIPTOR783 ...  
Instantiating slot CUISINE of frame MEAL-DESCRIPTOR783 ...  
--> The value is MEXICAN-CUISINE

...  
The meal descriptor is:  
The CUISINE is MEXICAN-CUISINE  
The EATING-CONFIG is STANDING  
The PRESENTATION is BUFFET

The FORMALITY is INFORMAL

JULIA goes on to its next task, choosing a main course, and suggests chili for the main course (we leave out the way it did that). At that point, it is reminded of a meal in which chili was served. If this meal had been a successful one, the reminding would provide suggestions about other parts of the meal. In this case, however, some guests, who did not eat hot food, were left unsatisfied.

\*\*\*\*\* Planning for the task MAIN-COURSE-TASK .....

TASK-TREE = (((<MAIN-COURSE-TASK> <APPETIZER-TASK> <SALAD-TASK>  
<DESSERT-TASK>) (<MEAL-TASK775>))

The active goal chain is ( SC-MAIN-COURSE <= M-MEAL )

<JULIA> What about CHILI791 for the main course?

USER => ok

<JULIA>

\*\*\*\*\* Reminded of MEAL80, where chili was the main course

\*\*\*\*\* Case-based reasoning with the case MEAL80

\*\*\*\*\* Frame #<FRAME 27067331> \*\*\*\*\*

Name: MEAL80

isa: (<M-MEAL>)

Category: INDIVIDUAL

Slots:

ACTUAL-RESULT: <ACTUAL-RESULT80>

CHARACTERS: (?HOST ?GUESTS ?PARTICIPANTS)

CONSTRAINTS: (<C-COST11>)

DEFINED-SLOTS: NIL

DESCRIPTOR: <MEAL-DESCRIPTOR80>

EXPECTED-RESULT: <EXPECTED-RESULT80>

FOLLOW-UP: NIL

GOALS: (<S-HUNGER80> <E-EAT80>)

GUESTS: (<\*JLK\*S-PARENTS>)

HOST: <\*JLK\*>

ORDER: ((<SC-SALAD80> <SC-MAIN-COURSE80>))

PARTICIPANTS: (?HOST ?GUESTS)

SETTING: <\*JLK\*S-HOUSE>

STEPS: (<SC-SALAD80> <SC-MAIN-COURSE80>)

\*\*\* Trying to do analogy-mapping the case MEAL80 ...

Checking if the previous plan for goals S-HUNGER80 E-EAT80 was successful .....

Previous plan execution failure found

\*\*\*\*\* Frame #<FRAME 27070771> \*\*\*\*\*

Name: ACTUAL-RESULT80

isa: (<ACTUAL-RESULT>)

Category: PROTOTYPE

Slots:

DEFINED-SLOTS: NIL

GOALS-ACHIEVED: NIL

GOALS-FAILED: (<S-HUNGER20> <E-EAT20>)

The set of goals failed was S-HUNGER80 E-EAT80

It was because ((NOT EVERY ONE ATE SPICY DISH))

JULIA will try to avoid making this mistake again. The previous failure was because of a missing constraint about spices, and JULIA seeks to find out if this constraint should be taken into account in the current case. JULIA therefore asks if any of the participants don't eat spicy food, finds out there are some, and creates a "non-spicy-food" constraint for the current case. It propagates that constraint and checks it against what it has already decided. It finds out that Mexican food is spicy. It therefore rules out Mexican as a cuisine, and because choosing a main course is dependent on having a value for cuisine, deletes the choose-a-main-course task from the task network, and reschedules the meal-descriptor task and the choose-a-main-course task. It then goes on from there.

\*\*\* Attempting to avoid the previous plan failure.....

The assigned blame was that C-NON-SPICY-PREF80 had not been considered.

To avoid previous plan failure ...

Asking the user of a missing constraint C-NON-SPICY-PREF

<JULIA> Is there anyone who doesn't like spicy food? (How many?)

USER => 3

<JULIA>

Trying to propagate the constraint C-NON-SPICY-PREF793 .....

--> Generating a new constraint C-NON-SPICY-CUISINE794

--> Generating a new constraint C-NON-SPICY-DISH795

Trying to propagate the constraint C-NON-SPICY-CUISINE794 .....

--> No constraint has been generated from C-NON-SPICY-CUISINE794

Trying to propagate the constraint C-NON-SPICY-DISH795 .....

--> No constraint has been generated from C-NON-SPICY-DISH795

Applying constraint C-NON-SPICY-PREF793

--> One of the characteristics of MEXICAN-CUISINE is spicy

--> Aborting MEXICAN-CUISINE

--> Killing the current task MAIN-COURSE-TASK .....

--> Rescheduling MEAL-DESCRIPTOR-TASK MAIN-COURSE-TASK into the task network .....

\*\*\*\* Planning for the task MEAL-DESCRIPTOR-TASK .....

TASK-TREE = ((<MEAL-DESCRIPTOR-TASK> <MAIN-COURSE-TASK> <APPETIZER-TASK>  
<SALAD-TASK> <DESSERT-TASK>) (<MEAL-TASK775>))

The active goal chain is ( MEAL-DESCRIPTOR <= M-MEAL )

Constraint Satisfaction .....

Checking constraints for slot CUISINE of MEAL-DESCRIPTOR783 ...

--> Applying constraint C-NON-SPICY-CUISINE794 to slot CUISINE

--> The slot CUISINE is not yet filled in

Instantiating slot CUISINE of name MEAL-DESCRIPTOR783 ...

Because there has been little decision making done until now, JULIA cannot suggest a new cuisine by itself at this point, so it asks the user again for a cuisine preference, this time telling the user constraints on the preference. The user suggests Italian, and JULIA goes on. Note that had the user originally stated her reason for choosing Mexican cuisine, it might have been possible for JULIA to suggest a new cuisine itself. For example, both Mexican and Italian cuisines make extensive use of cheese. JULIA will continue its reasoning, choosing lasagne for the main course (we will not show the reasoning), and will be reminded of a case in which vegetarians were at a lasagne dinner and could not eat (MEAL70). JULIA knows, however, that had the lasagne been meatless, they could have eaten. JULIA uses the process shown above to make that inference and goes on to plan the rest of the meal. JULIA finally suggests vegetarian antipasto as the appetizer, veggie lasagne and Italian bread for the main course, mixed green salad as the salad, and ice cream for dessert.

#### 4. Where's the learning?

What has been described in the previous pages is a problem solving or inference method in which previous cases rather than general knowledge are used during reasoning. Where, then, is the learning? We have two answers to this question. First, as a result of case-based reasoning, a reasoner improves its performance in several ways. It can anticipate

and avoid mistakes made previously. It can shortcut the process of making suggestions or hypotheses. Previous cases, in effect, cut down the search space when choices between alternatives must be made. Our second answer lies in the set of support processes we have not discussed in the paper. In order for case-based reasoning to work, a system must at least *learn by remembering*. Case-based reasoning uses cases that are remembered to the advantage of the problem solver. But, if the only learning method available was learning by remembering, a case-based problem solver would soon be swamped by the large number of cases it could remember. Thus, abstraction and generalization processes are necessary so that cases that fit the norm can be recognized and only those that differ be saved. This, of course, requires the whole combination of similarity and explanation based methods, both those based on success and those based on failure. There are also criteria for similarity that need to be revised as case-based reasoning is going on so that the reasoner won't repeat the mistakes it might make in selecting analogs. Case-based reasoning itself is the set of inference methods that capitalize on specific instances learned through remembering. To work efficiently, it requires the services of memory integration and retrieval processes and generalization and abstraction processes.

### **Acknowledgements**

This work is supported in part by NSF under Grant No. IST-8317711 and Grant No. IST-8608362, by ARO under Contract No. DAAG29-85-K-0023, and by ARI under Contract No. MDA-903-86-C-173. Much of what is in this paper came out of discussions with Nancy Carson, Rich Cullingford, Ellse Hill, Tom Hinrichs, Hong Shinn, and Roy Turner. Hong Shinn was responsible for the programming that went into the example above.

### **References**

- Alterman, R. (in press). The operational level of a commonsense planner. Submitted to the 1987 Cognitive Science Conference.
- Carbonell, J. G. (1983). Learning by analogy: Formulating and generalizing plans from past experience. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell, *Machine learning: An artificial intelligence approach*. Los Altos, CA: Morgan Kaufmann.
- Carbonell, J. G. (1986). Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell, *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos, CA: Morgan Kaufmann.
- Cullingford, R. E. & Kolodner, J. L. (1986). Interactive advice giving. In *Proceedings of the 1986 IEEE International Conference on Systems, Man, and Cybernetics* (pp. 709-714). Atlanta, GA: IEEE.
- Hammond, K. (1986). Learning to anticipate and avoid planning problems through the explanation of failures. In *Proceedings of AAAI-86* (pp. 556-560). Phila., PA: Morgan Kaufmann.
- Holyoak, K. J. (1985). The pragmatics of analogical transfer. In G. Bower (Ed.), *The psychology of learning and motivation* (Vol. 19). Orlando, FL: Academic Press.
- Kolodner, J. L. (1984). *Retrieval and organizational strategies in conceptual memory: A computer model*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Kolodner, J. L. (1985). *Experiential processes in natural problem solving* (Technical Report No. GIT-ICS-85/23). Atlanta, GA: Georgia Institute of Technology, School of Information and Computer Science.
- Kolodner, J. L. (1986). Some little-known complexities of case-based inference. In *Proceedings of The Third Annual Workshop on Theoretical Issues in Conceptual Information Processing* (pp. 61- 67). Phila., PA.
- Kolodner, J. L. (in press). Capitalizing on failure through case-based inference. Submitted to the 1987 Cognitive Science Conference.
- Kolodner, J. L. & Cullingford, R. E. (1986). Towards a memory architecture that supports reminding. In *Proceedings of the 1986 Conference of the Cognitive Science Society* (pp. 467-477). Amherst, Mass: Lawrence Erlbaum Associates.

- Kolodner, J. L., Simpson, R. L. & Sycara, K. (1985). A process model of case-based reasoning in problem solving. In *Proceedings of IJCAI-85* (pp. 284-290). Los Angeles, CA: Morgan Kaufmann.
- Lebowitz, M. (1983). Generalization from natural language text. *Cognitive Science*, 7, 1-40.
- Schank, R. C. (1982). *Dynamic memory*. Cambridge, UK: Cambridge University Press.
- Simpson, R. L. (1985). *A computer model of case-based reasoning in problem solving: An investigation in the domain of dispute mediation*. (Technical Report No. GIT-ICS-85/18). Doctoral dissertation, School of Info. and Comp. Science, Georgia Inst. of Technology, Atlanta, GA.
- Sycara, K. (1985). *Precedent-based reasoning in expert labor mediation* (Technical Report No. GIT-ICS-85-22). Atlanta, GA: Georgia Inst. of Technology, School of Info. and Comp. Science.
- Turner, R. (1986). A derivational approach to plan refinement for advice giving. In *Proceedings of the 1986 IEEE International Conference on Systems, Man & Cybernetics* (pp. 858-862). Atlanta, GA: IEEE.

# Appendix: Representing Procedural Information Declaratively

Roy M. Turner  
School of ICS  
Georgia Institute of Technology  
Atlanta, Georgia 30332

The controversy in artificial intelligence (AI) over the relative merits of declarative versus procedural representation for a program's knowledge would seem to have been resolved in favor of declarative representation. For example, rule-based expert systems' rules are declarative, as are MOLGEN's (Stefik, 1981) constraints, STRIPS (Fikes & Nilsson, 1971) plans and operators, and MDX's (Gomez & Chandrasekaran, 1982) specialists. Declarative representation offers many benefits, among them clarity, modularity, and the ability for both programmers and the program itself to examine and modify the program's knowledge.

One type of knowledge used by programs has seldom been represented declaratively, however: the knowledge that a program uses to manipulate its other knowledge. This knowledge can be thought of as comprising a program's *reasoning methods*. Examples of this sort of knowledge are the information used by rule-based systems to manipulate and apply their rules, the knowledge used by a planner in formulating a plan, the knowledge used by a case-based reasoner that tells it how to extract information from previous cases, or the information used by a diagnosis program such as INTERNIST-1 (Miller *et al.*, 1982) that allows it to form diagnoses.

It is at least as desirable to represent a program's reasoning methods declaratively as it is to represent its domain knowledge in an explicit form. With its reasoning methods in a declarative form, a program could use a relatively simple function or set of functions to *interpret* its reasoning methods. The reasoning methods, like the program's other knowledge, would be more clear to humans and easier for them to change, more modular, and would be represented in such a form that the program could (at least potentially) examine and change them to better fit the problems it is called upon to solve. In addition, a reasoner could have specializations of reasoning methods for special goals or goals arising in special situations; this would allow the reasoner to adapt its problem-solving behavior to the actual problem it is faced with.

I have begun at least a modest attempt towards representing a program's reasoning methods declaratively in the MEDIC project (Turner, 1988). MEDIC is a *schema-based* diagnostic reasoner which works in the domain of pulmonology. It represents its knowledge of how to solve diagnostic problems as packets of control knowledge called *schemata*<sup>1</sup> which it retrieves and applies in response to problem-solving goals.

In the remainder of this appendix, I will discuss some of the issues involved in representing procedural knowledge declaratively, then discuss schema-based reasoning.

---

<sup>1</sup>After (Lesgold *et al.*, 1981)'s term for the procedures used by human diagnosticians.

## Issues

In order to represent something, it is crucial that there exists a vocabulary rich enough to describe the thing to be represented; this is as true for representing reasoning methods as it is for representing domain knowledge. To represent reasoning methods, our vocabulary must be, at minimum, expressive enough to describe the reasoner's goals, the actions it can perform, and the order in which to perform the actions.

In addition to an appropriate vocabulary, a reasoner should have some means of representing abstractions of sequences of actions to perform. The advantage of this is that it effectively reduces the size of the space to be searched by the reasoner in order to find actions to achieve goals. This issue was addressed (for domain knowledge) by STRIPS, but especially by NOAH (Sacerdoti, 1977) and later problem reduction planners.

In any but the most trivial task, a reasoner will have many different pieces of knowledge—schemata—representing its reasoning methods. It will need to have some way of storing these schemata so that they can be retrieved efficiently and easily as they are needed. To do this, the reasoner should index its schemata by features of the situation(s) in which they are useful. A memory scheme such as described in (Kolodner, 1984) seems reasonable for this purpose.

One benefit of representing a program's reasoning methods declaratively is that there exists the potential for the reasoner to modify its existing schemata and learn new ones from experience. In order to foster this, the representation of the reasoning methods as schemata should facilitate machine learning. One approach to learning from experience is case-based reasoning. A possible representation for schemata, using this approach, is as generalized cases of problem-solving to achieve a goal—i.e., a schema would, on this view, represent the procedural information derived from cases of problem-solving.

## Schema-based Reasoning

Schema-based reasoning is one approach to representing a program's reasoning methods declaratively. In this section, I will briefly discuss schemata, their organization in and retrieval from memory, and how they are used in MEDIC.

### Schemata

A schema is a packet of procedural information that the reasoner can use to achieve a goal. For example, a medical reasoner would have schemata for guiding the course of a consultation, interpreting a finding, evaluating a hypothesis, etc. A schema contains information about its applicability, such as goals and features of problems for which it is useful, as well as containing information about the actions the reasoner should take in applying the schema. Figure 1 describes a schema to achieve the goal of interpreting a finding of dyspnea (shortness of breath); Figure 2 shows the Lisp code for the schema.

In MEDIC, a schema can have several parts:

1. goals,
2. a situation description,
3. actions, and
4. parameters.

Parameters: (finding)  
 Finding: a finding of dyspnea  
 Goals: Explain the finding  
 Steps:  
     elaborate: use `sc-elaborateDyspnea` to gather information  
                 about the finding  
     specialize: use `sc-specializeFinding` to try to specialize  
                 the finding (e.g., to dyspnea on exertion)  
     explain: assert two hypotheses to account for this finding:  
                 pulmonary disease and cardiac disease

Figure 1: Description of `sc-dyspnea`, schema to interpret dyspnea.

```

(defschema sc-dyspnea (~sc-finding)
  ;; parameter is "item", but make finding a synonym:
  (parms (item))
  (finding ?item)
  (goal
    (~a-explain (object ?finding)))
  (steps
    (start elaborate)
    (members
      (elaborate
        (goal (~a-elaborate (object ?finding)))
        (action
          (~sc-elaborateDyspnea
            (finding ?finding)))
        (next
          ((if t) (then (to specialize))))))
      (specialize
        (goal (~a-specialize (object ?finding)))
        (action (~specializeFinding
          (finding ?finding)))
        (next
          ((if :failure) (then (to explain)))
           ((if :success) (then (to elaborate))))))
      (explain
        (goal (~a-explain (object ?finding)))
        (action
          (~sc-explainDyspnea
            (finding ?finding)))
        (next
          ((if t) (then (done)))))))
  )
  )

```

Figure 2: `sc-dyspnea`—a schema for interpreting a finding of dyspnea.

Figure 2 shows some of these parts. The goals of a schema are those that the schema can achieve. For example, schema *sc-dyspnea* has the goal to explain a finding (“a-explain...”).<sup>2</sup>

In addition, a schema may provide information about the situation(s) in which it is useful. It does this via other slots, such as “patient” (not shown in Figure 2), which contain information about entities present in the problem. Information contained in these slots is used during retrieval to find appropriate schemata for a problem.

The actions of a schema are called *steps* (contained in the “steps” slot). Each step consists of three parts:

1. a description of a goal;
2. a description of an action to perform; and
3. information about what to do next (the “next information”).

The goal of a schema step, a subgoal of the schema, represents the goal that the step is designed to achieve. For example, in Figure 2 step “specialize” has the goal of specializing the finding of dyspnea.

The action portion of a step is either a primitive action (e.g., “specializeFinding” in step “specialize”) or a schema (e.g., “sc-elaborateDyspnea” in step “elaborate”). The primitive actions that MEDIC currently knows about are listed in Table 1; its schemata are listed in Table 2.

The “next information” contains information, referred to at run time, that allows the reasoner to select the next step based on the results of steps performed so far.<sup>3</sup> For example, step “specialize” in Figure 2 tells the reasoner to perform either step “explain” or step “elaborate” next, depending on whether or not “specialize” succeeded or failed. The “next information” is somewhat more expressive than that shown: values can be set in frames and values tested for in short-term memory, for example.

A schema can be used as a sub-schema of another schema; for this purpose, some schemata have a “parms” slot that tells the reasoner which slots to set as parameters to the schema (it is up to the person who enters the schemata, at this time, to make sure that the appropriate linkages exist between a schema and its subschemata).

Schemata are similar to traditional knowledge structures used to support reasoning. For example, if each step in the schema has an action that is a primitive action, then the schema is equivalent to a *script* (Schank and Abelson, 1977) with *tracks* (Cullingford, 1981). If, on the other hand, all of the steps of a schema have actions that are either schemata or primitive actions, the schema can be viewed as a abstract plan or as equivalent to one of NASL’s (McDermott, 1978) tasks. Finally, we can view a schema as a packet of rules, perhaps similar to MDX’s *concepts* or *specialists*. In this view, the rules’ “antecedents” would consist of the information in the schema—findings, characteristics of the patient, etc.—(excluding the schema’s steps); the “consequents” would be the actions that are specified by the schema.

There are several differences between schemata and these other knowledge structures, however. First, schemata are more general, in effect subsuming the functionality of the others. Second, new schemata can, in principle, be created by applying old schemata to new situations, then generalizing the result. This would allow schemata to be created for situations not anticipated when the reasoner was given its knowledge; the reasoner could adapt to its environment by operationalizing its

---

<sup>2</sup>“?” signifies a variable, which in our frame program, FrameWork (Turner, 1987), acts as a pointer; i.e., ?finding means “find the value of slot “finding” in the current frame.”

<sup>3</sup>In this sense, then, SBR represents a type of reactive planning (cf. Firby, 1987).

problem-solving knowledge. And third, schemata are active participants in memory organization, as will be discussed below. This organization allows the most specialized schemata for a portion of a problem to be retrieved and applied to that problem.<sup>4</sup>

So far, I have found three types of schemata to be useful for performing diagnosis:

1. global,
2. local, and
3. strategic.

A *global schema* is one that controls a large portion of a diagnostic session; it can be thought of as providing part of the overall framework of diagnosis. An example of a global schema is “sc-consult,” shown in Figure 3. This schema is used by the reasoner to guide a general consultation.

```
(defschema sc-consult (~sc-basic)
  (goal
    (^a-dx
      (actor ~*self*)
      (object $patient)))
  (steps
    (start getInitInfo)
    (members
      (getInitInfo
        (action
          (~sc-getInitInfo))
        (next
          ((if t) (then (to gatherInfo))))))
      (gatherInfo
        (action
          (~sc-gatherInfo))
        (next
          ((if t) (then (to formDx))))))
      (formDx
        (goal
          (^a-dx
            (actor ~*self*)
            (object $case)))
          (action
            (~sc-formDx))
          (next
            ((if t) (then (to handleFbk))))))
      (handleFbk
        (action
          (~sc-handleFbk))
        (next
          ((if t) (then (to updateMem))))))
      (updateMem
        (action
          (~sc-updateMem))
        (next
          ((if t)
            (then (done))))))))))
```

Figure 3: Global schema “sc-consult.”

---

<sup>4</sup>MDX’s specialists also participate in its memory organization; however, schemata are different from specialists both in the type of information they contain and in the way they are used by the reasoner.

*Local schemata*, on the other hand, control small portions of the diagnostic process relatively independent of the global structure of the consultation; they are activated to achieve relatively circumscribed goals. For example, *sc-dyspnea* (Figure 2) is a local schema which guides the interpretation of a finding of dyspnea.

A *strategic schema* represents a reasoning strategy. Strategic information is necessary because there may be several active schemata from which the reasoner needs to select the best one. For example, during a consultation, guided by *sc-consult*, the reasoner may encounter a finding of dyspnea, which activates *sc-dyspnea*. Should the reasoner continue applying *sc-consult*, temporarily ignoring the finding, or should it instead immediately follow up the finding? Information for making such decisions is represented in strategic schemata.

At the current time, my representation for strategic schemata is somewhat crude. Eventually, a vocabulary must be developed that will allow the reasoner to represent its internal state, including its goals and active schemata; such a vocabulary should also allow the representation of actions for the reasoner to take in selecting a schema to apply, much as schemata allow the representation of other sorts of actions.

A simple example of a strategic schema is shown in Figure 4. This schema provides a goal ordering for the reasoner that induces a crude form of hypothetico-deductive reasoning: select any goals (i.e., select their corresponding schemata) that relate to findings (in the hope of producing hypotheses); if there are none, then select goals that relate to hypotheses; if none, then select goal of gathering information from the user; and finally, if that cannot be done, select the goal of forming a diagnosis.

## Organization and Retrieval

In MEDIC, schemata are organized in a memory that is designed along lines similar to CYRUS (Kolodner, 1984). The memory is a set of interconnected discrimination nets in which the nodes are memory structures and the links are indices relating one memory structure to another. In MEDIC, there are basically two kinds of memory structures: schemata and dxMOPs. A dxMOP (diagnostic memory organization packet (cf. Schank, 1982; Kolodner and Kolodner, 1987)) represents a particular class of consultation—e.g., consultations involving pulmonary disease, or consultations in which the patient is an alcoholic. A diagram of a portion of MEDIC's memory is shown in Figure 5.

In MEDIC, schemata can be retrieved via two routes. First, as a problem unfolds, MEDIC keeps track of the "best" (i.e., most similar) dxMOP; this dxMOP represents a classification by MEDIC of the current problem. As new information arises during problem solving, MEDIC uses information contained in the dxMOP to determine what its goal should be with respect to the information, and, more importantly, it uses the dxMOP to determine what schema is generally useful for that goal in situations matching the current problem.

The second route to finding an appropriate schema is through specialization hierarchies of schemata. Schemata are active participants in memory, as mentioned above. Each schema indexes any specializations of itself. A schema which can achieve the goal of interpreting dyspnea, for example, would index other schemata for interpreting dyspnea on exertion, dyspnea on exertion in patients confined to wheelchairs, etc.

## Reasoning with Schemata

The general idea behind schema-based reasoning is that as goals arise during problem solving, the reasoner retrieves appropriate schemata from memory and applies them to achieve the goals.

```

(defconcept 'st-HDreas :isa ^strategy
:slots
'((actions
  (value
    (^set
      (members
        (
          ;; select finding schemata if present--or any
          ;; schemata that has a goal to do w/ a finding:
          ((test t)
            (action
              (choose
                (or
                  (goal (^a-explain (object (^finding))))
                  (goal (^a-interpret (object (^finding))))
                  (goal (^a-specialize (object (^finding))))
                  (goal (^goal (object (^finding)))))))
                ;; select hypotheses next:
                ((test t)
                  (action
                    (choose
                      (or (goal (^a-confirm (object
                        (^hypothesis))))
                        (goal (^a-verify (object
                          (^hypothesis))))
                        (goal (^a-specialize (object
                            (^hypothesis))))
                        (goal (^goal (object
                            (^hypothesis))))))))
                    ;; select whatever is trying to perform a gather
                    ;; information:
                    ((test t)
                      (action
                        (choose
                          (or (goal (^a-elaborate))
                              (and (goal (^a-know))
                                  (not (goal (^a-dx)))))))
                          ;; select whatever is trying to perform a
                          ;;diagnosis:
                          ((test t)
                            (action
                              (choose
                                (goal (^a-dx))))
                              ;; if nothing else, select first thing on agenda:
                              ))))))))
          ))))))

```

Figure 4: A simple strategy for hypothetico-deductive reasoning style.

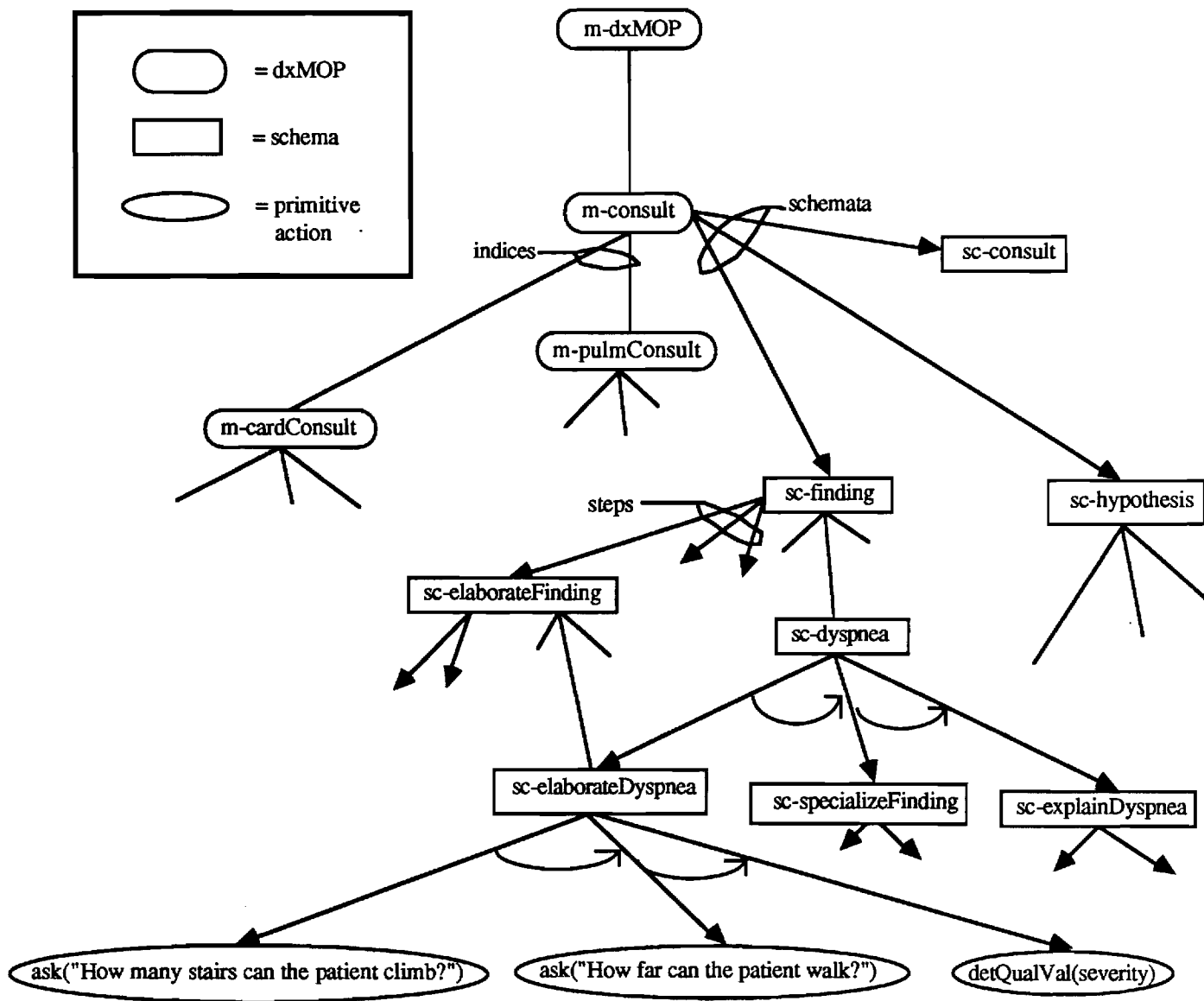


Figure 5: A portion of Medic's memory.

Applying a schema involves the following steps. First, a step is selected (using information contained in the schema). The step is then applied; if it is a primitive action, the reasoner performs the action; if it is a schema, then the schema is recursively applied; and if there is no action, or if the action failed, then the reasoner attempts to find and apply other schemata or primitive actions that can achieve the step's goal. A new step is then selected, using information contained in the current step's "next information," and application continues until the schema has been fully applied.

Often there should be more than one schema active at a time. Goals may arise at any time during problem solving; e.g., asking a question may provide information about an unsuspected finding, which gives rise to the goal to interpret that finding. When a goal arises, the reasoner should interrupt what it is doing to find and activate a schema for the goal. It can then select the best schema to pursue, given its active schemata and goals and the characteristics of the problem.

The problem of selecting an appropriate schema at each point in problem solving is one of attention focusing: what goal should the reasoner pursue? In order to focus the reasoner's attention when there is more than one active schema, we use the idea of meta-reasoning (e.g., Davis & Buchanan, 1984): reasoning that takes place to guide planning. In our approach, meta-reasoning information is present in our strategic schemata.

## Conclusion

Just as declarative representation of domain knowledge offers benefits to a reasoner, declaratively representing the reasoning methods used to manipulate that knowledge is advantageous. By representing reasoning methods in a declarative form, a reasoner's procedural knowledge is made more clear, becomes easier to modify by humans, is more modular, and allows the reasoner to examine and modify its knowledge of how to solve problems.

Schema-based reasoning uses schemata to represent procedural information declaratively. It allows the reasoner to respond to goals as they arise during problem solving by finding and applying the appropriate piece(es) of procedural knowledge. Though I have so far only examined the use of schemata in diagnostic reasoning, schema-based reasoning should prove useful in many other problem-solving tasks.

**Table 1: Medic's primitive actions.**

---

**ACCOUNTFORLCS:** Mark findings as accounted for by a Logical Competitor Set.  
**ADDAGENDA:** Add a schema to the agenda.  
**ANSWERFROMDXMOP:** Find the answer to a question from a dxMOP.  
**ANSWERFROMSTM:** Find the answer to a question from short-term memory.  
**APPENDVALUE:** Append a value onto a slot.  
**ASK:** Ask a question.  
**ASK-AND-PARSE:** Ask a question and parse the result.  
**ASKQUESTION:** Ask a "question" frame (a prediction).  
**ASSERT:** Assert something in STM.  
**CLEAR:** Clear the window panes.  
**COMPUTE:** Compute the value of an expression.  
**CREATEFINDING:** Create a finding, adding it to STM.  
**DEFAULTHYPQUESTIONS:** Use semantic representation of a hypothesis to generate questions to ask (predictions).  
**DETQUALVAL:** Perform actions to determine the qualitative value of some slot.  
**EXPLAINDEFAULT:** Explain a finding using information contained in the finding's definition.  
**FINDINCOMPLETESLOT:** Find a slot that is unfilled in a frame.  
**FORMLCS:** Form an LCS.  
**GOALSATISFIED?:** Return "t" if a goal is satisfied.  
**INITMEDIC:** Initialize the system.  
**MULTIPLECBI:** Perform multiple case-based inferences (unimplemented).  
**POPVALUE:** Pop a value from a slot, returning it to the parent schema.  
**PRESENTDX:** Present a diagnosis to the user.  
**PRINT:** Print something on the user interface.  
**RATECONFIDENCE:** Convert a score into a confidence value.  
**RELATETOSTM:** Relate a hypothesis to other things in STM.  
**RETRIEVE:** Retrieve a concept from memory based on a pattern.  
**RUNWAITING:** Run waiting schemata.  
**SCOREEXPLAINED:** Score a hypothesis by what it explains.  
**SCOREFAILED:** Score a hypothesis by its failed predictions.  
**SCORERELATED:** Score a hypothesis by other hypotheses it is related to (unimplemented).  
**SCOREUNEXPLAINED:** Score a hypothesis based on what it doesn't explain.  
**SELECTHYPPRED:** Select a prediction to ask about.  
**SELECTLCSQUESTION:** Select a question from an LCS to ask about; aim is to solve the LCS (like INTERNIST-1).  
**SETUPCONSULTSTM:** Set up the STM for a consultation.  
**SETVALUE:** Set a value in some slot.  
**SPECIALIZEFINDING:** Specialize a finding based on what is known about it.  
**SPECIALIZEHYP:** Specialize a hypothesis based on what is known about it.  
**TEST:** Perform some arbitrary boolean test.  
**UPDATEFINDING:** Update a finding based on information from the user.

**UPDATEFROMSTM:** Update a hypothesis based on information in STM.

**UPDATEPATIENT:** Update information about the patient based on information from the user.

**UPDATESA:** Update the dxMOP and strategy in use.

**WAIT:** Wait for some condition to be met, then continue.

---

**Table 2: Medic's schemata.**

---

SC-ANSWERHYPQUESTIONS: Answers hypothesis' questions.  
SC-ANSWERQUESTION: Answer a question.  
SC-BASIC: Basic diagnostic schema; not useful itself, but organizes others.  
SC-CBR: Perform case-based reasoning.  
SC-CONSULT: Schema for performing a consultation.  
SC-DYSPNEA: Interpret finding of dyspnea.  
SC-ELABORATEDYSPNEA: Elaborate a finding of dyspnea.  
SC-ELABORATEFINDING: Elaborate finding.  
SC-EVALUATEHYP: Evaluate a disease hypothesis  
SC-EXPLAINDYSPNEA: Explain dyspnea.  
SC-EXPLAINFINDING: Explain a finding.  
SC-FINDDYSPNEAONSET: Find under what conditions dyspnea occurs.  
SC-FINDDYSPNEASEVERITY: Determine the severity of dyspnea.  
SC-FINDINFO: Find a piece of information.  
SC-FINDING: Interpret a finding.  
SC-FINDINGSLOTVALUE: Find a slot value for a finding.  
SC-FORMDX: Form a diagnosis.  
SC-GATHERINFO: General purpose information-gathering schema.  
SC-GENERATEHYPQUESTIONS: Generate a set of 'questions' to ask about the hypothesis.  
SC-GETINITINFO: Get patient description, chief complaint.  
SC-HYPOTHESIS: Evaluate and specialize a disease hypothesis.  
SC-INITCONSULT: Initialize system for new consultation.  
SC-PROCESSINFO: Process information obtained from the user.  
SC-RELATEHYP: Relate a hypothesis to others.  
SC-RESTING: "Resting" schema.  
SC-SATISFYGOAL: Satisfy a goal.  
SC-SCOREHYPOTHESIS: Score a hypothesis.  
SC-SPECIALIZEFINDING: Specialize a finding.  
STM-FINDING: Interpret information about a finding, adding it to STM.  
STM-PATIENT: Interpret information about the patient, adding it to STM.  
STM-SCHEMA: Interpret information, possibly adding to STM.

---

## References

- Cullingford, R.E. (1981). SAM. In *Inside Computer Understanding*, R.C. Schank and C.K. Riesbeck (Eds.), Hillsdale, NJ: Lawrence Erlbaum Associates.
- Davis, R., and Buchanan, B.G. (1984). Meta-level knowledge. In B.G. Buchanan and E.H. Shortliffe (eds.), *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley Publishing Company, Reading, Massachusetts, pp. 507-530.
- Fikes, R.E., and Nilsson, N.J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving, *Artificial Intelligence*, v. 2.
- Firby, R.J. (1987). An investigation into reactive planning in complex domains, in *Proceedings of the Sixth National Conference on Artificial Intelligence*, pp. 202-206.
- Gomez, F., and Chandrasekaran, B. (1982). Knowledge organization and distribution for medical diagnosis. In W.J. Clancey and E.H. Shortliffe (Eds.), *Readings in Medical Artificial Intelligence*, pp. 320-338. Reading, Massachusetts: Addison-Wesley Publishing Company, 1984. (Originally published in *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-11, No. 1, pp. 34-42 (1981).)
- Kolodner, J.L. (1984). *Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model*, Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey.
- Kolodner, J.L., and Kolodner, R.M. (1987). Using experience in clinical problem solving: Introduction and framework. In *Proceedings of the 1987 IEEE International Conference on Systems, Man, and Cybernetics*.
- Lesgold, A.M., Feltovich, P.J., Glaser, R., and Wang, Y. (1981). The acquisition of perceptual diagnostic skill in radiology. Technical report No. PDS-1, University of Pittsburgh Learning Research and Development Center.
- McDermott, D. (1978). Planning and acting, *Cognitive Science*, vol. 2, pp. 71-109.
- Miller, R.A., Pople, H.E., Jr., and Myers, J.D. (1982). INTERNIST-1, an experimental computer-based diagnostic consultant for general internal medicine, *New England Journal of Medicine*, vol. 307, pp. 468-476.
- Sacerdoti, E.D. (1977). *A Structure for Plans and Behavior*, Elsevier, New York.
- Schank, R.C. (1982). *Dynamic Memory*, Cambridge University Press, New York.
- Schank, R.C., and Abelson, R. (1977). *Scripts, Plans, Goals and Understanding*, Lawrence Erlbaum Associates, Hillsdale, NJ.
- Stefik, M. (1981). Planning with constraints, MOLGEN: Part 1, *Artificial Intelligence*, v. 16 #2, pp. 111-139.
- Turner, R.M. (1987a). *Issues in the Design of Advisory Systems: the Consumer-Advisor System*, Technical Report #GIT-ICS-87/19, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA 30332.
- Turner, R.M. (1988). Organizing and using schematic knowledge for medical diagnosis, to appear in the Proceedings of the DARPA Workshop on Case-based Reasoning, Clearwater Beach, Florida.

PUBLICATIONS RESULTING FROM RESEARCH UNDER IST-8317711

- Kolodner, J.L. (1985). "Experiential Processes in Natural Problem Solving." Technical Report No. GIT-ICS-85/23. School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA.
- Kolodner, J.L. (1985). "'If you want my advice....': Some Protocols of a memory-intensive task," in "Proceedings of the Second Annual Workshop on Theoretical Issues in Conceptual Information Processing," New Haven, CT, May, 1985.
- Kolodner, J.L. and Simpson, R.L. (1985). "Machine Learning Research at Georgia Tech: Using Experience as a Guide to Problem Solving." Published in the "Proceedings of the Machine Learning Workshop," Skytop, PA, June 24-26, 1985, pp. 96-99.
- Kolodner, J.L. (1985). "Memory for Experience." In Bower, G., ed., The Psychology of Learning and Motivation, Academic Press, New York, 1985, pp. 1-57. Also Technical Report No. GIT-ICS-84/23. School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA.
- Kolodner, J.L., Simpson, R.L. and Sycara, E. (1985). "A Process Model of Case-Based Reasoning in Problem Solving." Published in the "Proceedings of the Seventh International Joint Conference on Artificial Intelligence," Los Angeles, CA Aug., 1985, pp. 284-290. Also Technical Report No. GIT-ICS-85/01. School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA.
- Kolodner, J.L. and Cullingford, R.E., "Interactive Advice-Giving." Published in the "Proceedings of the IEEE International Conference on Systems, Man and Cybernetics," Atlanta, GA, Oct., 1986. Also Technical Report No. GIT-ICS-86/20. School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA.
- Kolodner, J.L. and Simpson, R.L. (1986). "Problem Solving and Dynamic Memory," (long version). In Kolodner, J.L. and Riesbeck, C.K. Experience, Memory and Reasoning, Lawrence Erlbaum Assoc., Hillsdale, NJ. Technical Report No. GIT-ICS-84/24. School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA.
- Kolodner, J.L. (1986). "Some Unknown Complexities of Case-Based Inference." Presented at conference on "Theoretical Issues and Conceptual Information Processing" (TICIP), Atlanta, GA, Nov., 1986.
- Kolodner, J.L. and Cullingford, R.E. "Towards a Memory Architecture that Supports Reminding." Published in the "Proceedings of the Eighth Annual Conference of the Cognitive Science Society," Amherst, MA, Aug., 1986. Also Technical Report No. GIT-ICS-86/10. School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA.
- Kolodner, J.L. (1986). "Using Experience as a Guide for Problem Solving in Machine Learning: A Guide to Current Research," Kluwer Academic Press, 1986.
- Kolodner, J.L. "Capitalizing on Failure through Case-Based Inference." Published in the "Proceedings of the Ninth Annual Conference of the Cognitive Science Society," July, 1987.

- Kolodner, J.L. (1987). "Case-Based Inference: A Collection of Papers." Technical Report No. GIT-ICS-87/18. School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA.
- Kolodner, J.L. "Extending Problem Solver Capabilities through Case-Based Inference." Published in the "Proceedings of the Fourth International Machine Learning Workshop," June, 1987.
- Kolodner, J.L. and Kolodner, R.M. (1987). "Using Experience in Clinical Problem Solving: Introduction and Framework." Published in "IEEE Transactions on Systems, Man, and Cybernetics." Technical Report No. GIT-ICS-85/21. School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA.
- Kolodner, J.L. and Simpson, R.L. (1988). "The MEDIATOR: A Case Study of a Case-Based Problem Solver." Technical Report No. GIT-ICS-88/11. School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA.
- Turner, R.M. (1988a). "Organizing and Using Schematic Knowledge for Medical Diagnosis." Presented at DARPA Workshop on Case-Based Reasoning, May 11-13, 1988, Clearwater Beach, FL.
- Turner, R.M. (1988b). "Using Schemata for Diagnosis." Submitted to the Twelfth Annual Symposium on Computer Applications in Medical Care (SCAMC-88).
- Turner, R.M. (1988c). "Opportunistic Use of Schemata for Medical Diagnosis." Submitted to the Tenth Annual Conference of the Cognitive Science Society, Aug., 1988.

GIT-ICS-85/18

A COMPUTER MODEL OF CASE-BASED REASONING  
IN PROBLEM SOLVING:

An Investigation in the Domain of  
Dispute Mediation

Robert L. Simpson, Jr.

June, 1985

School of Information and Computer Science  
Georgia Institute of Technology  
Atlanta, Georgia 30332

This research has been supported in part by the Air Force  
Institute of Technology, in part by NSF grants #IST-8116892  
and #IST-8317711, and in part by the Army Research Office  
through grant #DAAG 29-85-K-0023.

ABSTRACT

Rather than approach each problem as a unique event, people often try to solve problems by recalling similar previous experiences as guides to problem solving. This analogical process, which we call case-based reasoning, seems to provide an explanation for the change in problem solving behavior of people over time. This research presents a computer process model of problem solving based on the use of case-based reasoning. The necessary reasoning processes, operational measures of similarity, and memory structures needed for effective storage and retrieval are presented via the specifications for an advisory system called the MEDIATOR, which offers advice on resolving common sense disputes. In this context, issues associated with enabling machines to dynamically adapt their reasoning and automatically recover from failure are discussed. The model of case-based problem solving which has been developed seems to offer promise as an integrated solution for some issues in problem solving, analogical reasoning, and machine learning.