A Model-Based Approach to Case Adaptation

Ashok K. Goel*

College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332

Abstract

In case-based reasoning, a given problem is solved by adapting the solutions to similar problems encountered in the past. A major task in case-based problem solving is to generate modifications that are useful for adapting a previous solution to solve the present problem. We describe a model-based method that uses qualitative models of cases for generating useful modifications. The qualitative model of a case expresses a problem-solver's comprehension of how the solution satisfies the constraints of the problem. We illustrate the model-based method in the context of case-based design of physical devices. A designer's understanding of how the structure of a previously encountered design produces its functions is expressed in the form of a function-structure model. The functional differences between a given problem and a specific case are mapped into structural modifications by a family of modification-generation plans. Each plan is applicable to a specific type of functional difference, and uses the function-structure model to identify the specific components that need to be modified. We discuss the evaluation of this model-based method in an experimental case-based system called KRITIK.

Case Adaptation in Case-Based Reasoning

Much of real-world problem solving appears to be case-based. Cognitive agents often seem to solve new problems by retrieving and adapting solutions to similar problems that they have encountered in the past. These observations have recently led to the development of several computational models of case-based reasoning [Ashley and Rissland 1988] [Hammond 1989] [Kolodner and Simpson 1990] [Riesbeck and Schank 1989]. These computational models posit different methods for adapting previous cases for solving new problems. For example, *heuristic search* [Stallman and Sussman 1977], *heuristic association* [Hammond 1989], and *derivational analogy* [Carbonell 1986]. The case-based method itself has been recursively used to adapt cases [Kolodner and Simpson 1990].

Methods for case adaptation differ in the types of knowledge, inference, and control they use. The method of heuristic-association, for example, uses knowledge of situation-specific associations that directly map differences between the specifications of the new and the known problems into modifications to the solution of the known problem. In addition, case-adaptation methods differ in the types of knowledge stored in the cases. In the method of derivational analogy, for example, a case contains a specification of a problem, a specification of a solution to the problem, and a specification of the problem-solving process by which the cognitive agent arrived at the solution.

In this paper, we describe another computational model for case-based reasoning in which *qualitative models of cases* are used to generate case modifications.

A Model-Based Method for Case Adaptation

The core idea in the model-based approach to case adaptation is that in addition to their knowledge of solutions to problems encountered previously, cognitive agents often also *comprehend* how a solution actually satisfies the constraints of the problem. This comprehension can be expressed in the form of a qualitative model for the case and used for adapting the case in solving a new problem. A designer, for example, may not only know the functions and the structure of a previously encountered artifact, but may also comprehend how the structure of the artifact produces its functions. If and when this type of knowledge is available in memory, it can be accessed and used in adapting the structure of the known artifact to design new ones.

The main issues in this model-based approach to case adaptation, then, are (i) how to represent the qualitative models of cases, (ii) how to index the models in memory, (iii) how to access them when needed, and (iv) how to use them in case adaptation. The KRITIK project investigates these issues in the context of solving design

*Some of this work was conducted when the author was with the Laboratory for Artificial Intelligence Research at the Ohio State University. There this work was supported by the Defense Advanced Research Projects Agency, contract F-49620-89-C-0110, and the Air Force Office of Scientific Research, grant 89-0250. At Georgia Institute of Technology, this work has been partially supported by the Defense Advanced Research Projects Agency, contract F-49620-88-C-0058.

problems in the domain of physical devices [Goel 1989]. KRITIK is a fully operational system that uses the case-based method for designing physical devices, and the model-based method for adapting design cases. It thus integrates model-based reasoning with case-based reasoning.

In KRITIK, the case-adaptation task is to map the differences between the function desired of a device and the function delivered by the retrieved design case into candidate modifications to the structure of the known design (*functional differences → structural modifications*). KRITIK uses knowledge of how the structure of the known device achieves its functions (*structure → function*) for generating useful candidate modifications. This knowledge is expressed as a qualitative *function-structure model* that specifies how the internal causal mechanisms of the known device compose the functional abstractions of its structural components into the functions of the device as a whole.

A design case in KRITIK contains three types of knowledge: (i) a specification of the functions delivered by a previously encountered design, (ii) a specification of the structure of the stored design, and (iii) a pointer to the qualitative function-structure model of the design. The cases are indexed by the functions delivered by the stored designs, and themselves act as indices to the function-structure-models of the designs. The knowledge content of a given function-structure model is specific to a particular design case but the representation language is useful for specifying the models of a large class of physical devices.

KRITIK uses *a family of modification-generation plans* for mapping the differences between the function desired of and delivered by the retrived design case into candidate modifications to the structure of the known design. These plans are indexed by the *types of functional differences* to which they are applicable. Each plan knows of the *types of structure modifications* that can help to reduce a specific functional difference but does not know what components in the specific design case need to modified. The modification-generation plans access the case-specific qualitative model, and use it to identify the specific components that need to be modified.

An Illustrative Example from KRITIK

Let us consider, as a simple illustrative example from KRITIK, the task of designing a device to cool high-acidity Sulfuric Acid by the case-based method. Let us assume that a number of previously encountered designs are available in memory, and that the case-retrieval task results in retrieving the design of a Nitric Acid Cooler (NAC) that cools low-acidity Nitric Acid. The design of NAC is shown in Figure 1.

The desired function and the delivered function in this example differ in (i) the substance to be cooled (Sulfuric Acid instead of Nitric Acid), and (ii) a property of the substance (high-acidity instead of low-acidity). The task of case adaptation is to map these functional differences into modifications to the structure of NAC such that the modified design can deliver the desired function of cooling high-acidity Sulfuric Acid. The task of modification generation is to propose candidate modifications that can help to deliver the desired function.

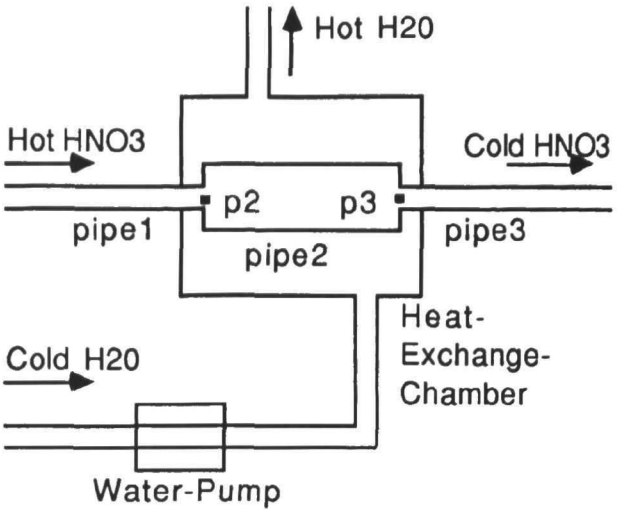

Figure 1: The Nitric Acid Cooler

Model-Based Case Adaptation: Knowledge, Inference, and Control

The generation of useful candidate structural modifications is in general computationally complex because (i) the differences between the function desired of and delivered by the retrieved design case can be large and many, (ii) the needed structural modifications can be non-local and many, (iii) there may be no simple correspondence between the functional differences and the structural modifications, and (iv) the structural modifications can interact with one another and with the components in the structure of the known design. Therefore, solving the modification-generation task computationally efficiently and effectively in general requires knowledge that can help to constrain and focus the process of generating useful modifications.

KRITIK uses three types of knowledge for adapting design cases:

1. Typologies of Functional Differences and Structural Modifications: The typology of functional differences classifies differences between the functions desired of and delivered by a design case; the typology of structural modifications similarly classifies modifications to the structure of a design.

2. Case-Specific Function-Structure Models: These models describe how the solution in a design case (the design structure) satisfies the constraints of the design (the functions of the design).

3. A Family of Modification-Generation Plans: These plans specify compiled sequences of abstract opera-

tions for mapping functional differences into candidate structure modifications.

Given a specific difference between the function desired of (e.g., to cool high-acidity Sulfuric Acid) and delivered by (e.g., to cool low-acidity Nitric Acid) a design case, KRITIK sets up four subtasks of the modification-generation task:

(i) Plan Selection: First, the functional difference is used as a probe into the functionally-indexed memory of modification-generation plans to select the applicable plan.

(ii) Plan Instantiation: Next, the retrieved plan is instantiated in the context of the specific design case to be adapted.

(iii) Model Retrieval: Then, the instantiated plan uses the case as a pointer to retrieve the function-structure model for the design case.

(iv) Plan Execution: Finally, the plan uses the case-specific function-structure model to generate candidate structure modifications that can help to achieve the desired function.

A Function-Structure Model

A case-specific function-structure model in KRITIK is an instantiation of a more general *component-substance model* of the functioning of a large class of physical devices. The component-substance model extends and generalizes the component and substance ontology used earlier in the *consolidation method* for deriving the behaviors of a device from the behavioral interactions between its structural components [Bylander and Chandrasekaran 1985]. Knowledge of a case-specific function-structure model in KRITIK is represented and organized in a *behavioral representation language*. The behavioral representation language extends and generalizes the *functional representation scheme* for representing knowledge of the functioning of a device [Sembugamoorthy and Chandrasekaran 1986].

The function-structure model of a given design case in KRITIK explicitly represents the structure, the functions, and the internal causal behaviors of the design. The behaviors compose the structural and behavioral interactions between the structural components into the functions of the device as a whole.

Structure: The structure of a device is viewed as constituted of components (e.g., battery, pipe), substances (e.g., water, electrical charge), and structural relations between them (e.g., connection, containment). The substances can be abstract, e.g., heat. The components and substances can have behavioral interactions. For example, substances can flow from one component to another if there is a certain type of structural relation between the two components, viz., the connection relation. The model borrows a typology of behavioral

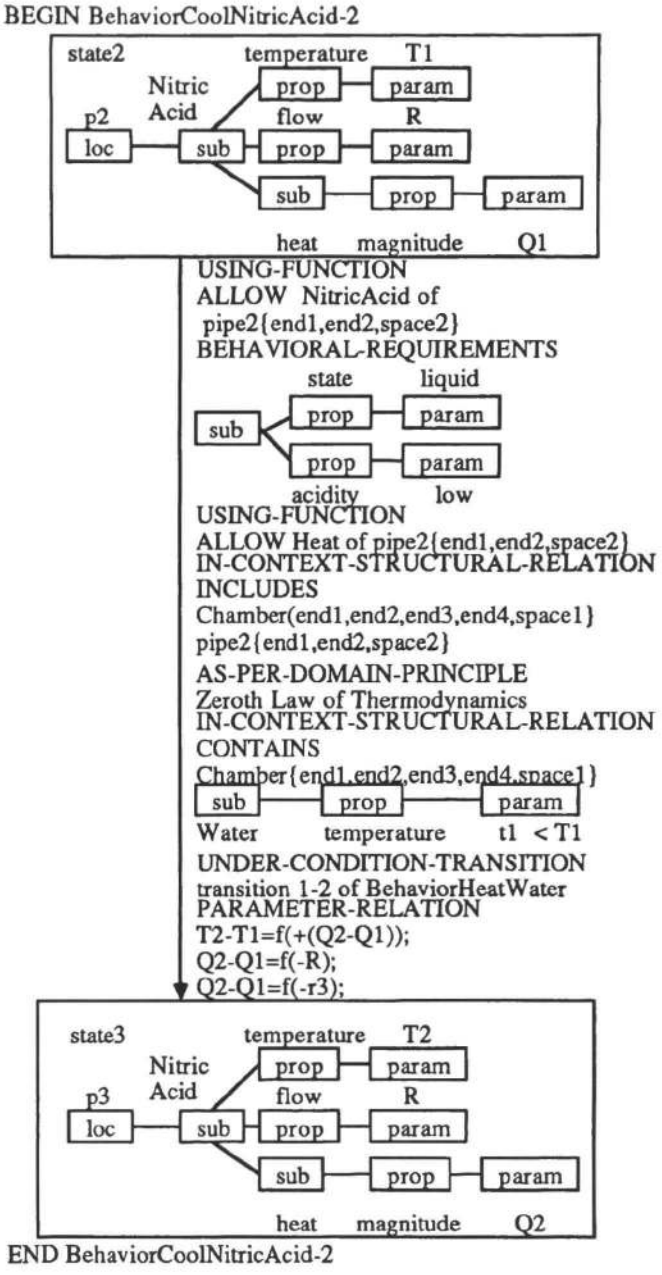

Figure 2: Behavior CoolNitricAcid-2

interactions from the consolidation method. For example, a battery *pumps* electrical charge and a pipe *allows* substances with certain properties. Knowledge of the structure of the device is organized in a structure-substructure hierarchy. A substructure is represented as a schema that specifies its functional abstractions, structural relations, modalities, parameters, etc.

Functions: Knowledge of the functions of a device is also represented in the form of schemas. The schema for function specifies the behavioral state it takes as input and the behavioral state it gives as output. It also specifies the internal causal behaviors responsible for transforming the input behavioral state into the output behavioral state. Thus, like in the functional representation scheme, the functions act as indices to the behaviors. In addition, the function schema specifies the device conditions under which the behavior accomplishes the function, and the stimulus from the environment which triggers the behavior.

Internal Causal Behaviors: Knowledge of the internal causal behaviors of a device is represented as directed acyclic graphs (DAGS) of behavioral states and state transitions. A fragment of one behavior of NAC is shown in Figure 2. A behavioral state in an internal behavior is represented in the form of schemas. The state schema specifies the location, property, and parameters and parameter values of a substance. For example, the schema labeled *state*2 in Figure 2 specifies that (some quantity of) Nitric Acid is at point $p2$ in the device space, and has the properties of temperature and flow rate with values $T1$ and R respectively. The state schema may also specify the substances contained within a substance, e.g., in *state*2 in Figure 2, Nitric Acid contains heat with magnitude $Q1$.

A behavioral-state transition in an internal behavior is annotated by the behavioral interactions that cause the transition to occur, e.g., the transition *state*2 $\Rightarrow$ *state*3 in Figure 2 is caused (among other causes) by the function *allow* of *pipe*2, where *pipe*2 *allows* the flow of low-acidity liquids. A state transition may also be annotated by the enabling conditions under which the behavioral interactions result in the transition. Some of the enabling conditions may pertain to structural relations between components, e.g., the structural relation that the *heat-exchange-chamber includes pipe*2 in transition *state*2 $\Rightarrow$ *state*3 (see Figure 1). Also, the enabling conditions in one behavior may act as indices to another other behavior, e.g., the enabling condition *under−state−transition* in transition *state*2 $\Rightarrow$ *state*3 refers to another behavior of NAC (BehaviorHeatWater, not shown here). In addition, a transition may be annotated by knowledge of deeper domain principles and qualitative equations as indicated in Figure 2.

Functional Differences and Structure Modifications

The component-substance model provides a vocabulary for expressing (i) certain types functional differences between design cases, and (ii) certain types of modifications to the structure of a design. The typology of functional differences includes the categories of substance difference, substance property difference, substance location difference, component difference, component modality difference, and component parameter difference. The typology of structure modifications includes the categories of substance substitution (including substance generalization and specialization), component modification (including component replacement, component modality change, and component parameter adjustment), relation modification (such as series-to-parallel and parallel-to-series conversion), substructure deletion (such as component deletion), and substructure insertion (such as substructure replication).

Given a function desired of a design, (e.g., to cool high-acidity Sulfuric Acid) and the function delivered by a specific design case (e.g., to cool low-acidity Nitric Acid), KRITIK classifies the differences between the two functions according to its typology of functional differences. If the desired and the delivered functions differ in more than feature, then it heuristically ranks the differences in order of the difficulty of reducing them. In the NAC example, for instance, the desired function and the delivered function differ in two features: *substance*1 $\rightarrow$ *substance*2 (Nitric Acid $\rightarrow$ Sulfuric Acid), and *property*1 $\rightarrow$ *property*2 (low-acidity $\rightarrow$ high-acidity). Since, in the domain of physical devices that can be modeled in terms of flow of substances between components, reducing the difference *substance*1 $\rightarrow$ *substance*2 is in general less difficult than reducing *property*1 $\rightarrow$ *property*2, KRITIK reduces the latter before the former.

A Family of Modification-Generation Plans

The types of knowledge, inference, and control, needed for reducing different types of functional differences are in general different. This implies a *family* of plans for generating structural modifications, where each plan in this family is applicable to a specific type of functional difference. For example, the *substance-property-difference plan* is useful for reducing the difference in the property of a substance, e.g., low-acidity $\rightarrow$ high-acidity. KRITIK uses modification-generation plans for several different types of functional differences. These plans are indexed by the type of functional differences they can help to reduce. The substance-property-difference plan, for example, is indexed by the functional difference *property*1 $\rightarrow$ *property*2. Each plan knows of the types of structure modifications that can help to reduce a specific functional difference. The substance-property-difference plan, for example, knows that a difference in the property of a substance can potentially be reduced by the structural modifications of (i) component parameter adjustment, (ii) component modality change, and (iii) component replacement.

The modification-generation plans, however, do not know what components in the given design need to modified. The substance-property-difference plan, for example, does not know which components in NAC's design need to be modified. This is determined by the component-substance model for NAC. First, the substance-property-difference plan uses the NAC case as a pointer to retrieve the component-substance model for NAC (recall that a case contains a pointer to its qualitative model). Next, since the substance property difference *low-acidity* → *high-acidity* occurs in the function of cooling low-acidity Nitric Acid, the substance-property-difference plan uses this function to access the internal causal behavior responsible for it, a fragment of which is shown in Figure 2 (recall that in the component-substance model, functions act as indices to the behaviors responsible for them). Then, the substance-property-difference plan traces through the retrieved behavior, checking each state transition in it to determine whether reducing the substance property difference *low-acidity* → *high-acidity* requires any component in the transition to be modified (or replaced). If so, it generates the corresponding structure modification. For example, when the plan arrives at the transition *state*2 ⇒ *state*3 shown in Figure 2, it finds that *pipe*2 allows the flow of only low-acidity substances. It therefore generates the structure modifications of (i) component parameter adjustment (in case *pipe*2 can allow the flow of high-acidity substances in a different parameter setting), (ii) component modality change (in case *pipe*2 can allow the flow of high-acidity substances in a different mode of operation), and (iii) component replacement (in case the first two modifications are not possible and *pipe*2 has to be replaced with some *new* – *pipe*2 which can allow the flow of high-acidity substances). The generated modifications can now be evaluated but that is different task altogether (see Goel [1989] for how KRITIK uses the model-based approach to solve it).

In this way, the substance-property-difference plan uses the component-substance model for NAC to generate structural modifications that can help to reduce the functional difference *low-acidity* → *high-acidity*. The substance-difference plan corresponding to the functional difference of *substance*1 → *substance*2 (Nitric Acid → Sulfuric Acid) similarly results in the generation of the structure modification of substance substitution $NitricAcid \rightarrow SulfuricAcid$.

Evaluation of the Model-Based Method

The KRITIK system evaluates model-based method for case adaptation in two different domains: heat exchangers such as NAC, and electrical circuits such as the electrical circuit in a flashlight. This insures that the method is not specific to any narrow domain. The design tasks KRITIK solves range from simple "naive" design tasks in the domain of electrical circuits to complex "expert" tasks in the domain of heat exchangers. We are currently extending the KRITIK system to two new domains: electromagnetic devices such as buzzers on house doors, and rotational devices such as reaction wheels.

KRITIK shows that the model-based method is quite effective for adapting design cases because the case-specific component-substance model explicitly represents the behavioral states of the known device, the functional role played by each structural component in the state transitions, the enabling conditions for the transitions, and so on. It also shows that the model-based method is quite efficient because each modification-generation plan needs to search only a small portion of the design. For example, the substance-property-difference plan searches only the internal causal behavior responsible for the function of cooling low-acidity Nitric Acid (ignoring all structural components that do not play any functional role in this behavior). The organization of the component-substance model thus constrains and focuses the search for generating structure modifications that can help to reduce a given functional difference.

Of course, the model-based method, like any other method, is applicable only when the types of knowledge it uses are available in the domain of interest. Our experience with KRITIK suggests that case-specific component-substance models are readily available in the domain of physical devices. Whether the model-based method is useful outside physical domains is an empirical issue that KRITIK does not answer. In any event, the domain of physical devices is very large and important; it includes, for example, all engineering devices.

Related Research

Goel and Chandrasekaran [1989] have earlier proposed the use of function-structure models for generating modifications to the designs of physical devices for achieving new device functionalities. The KRITIK system transforms their proposal into a computational model for case-based problem solving.

Simmons and Davis [1987] have used causal domain models for debugging plans but only for testing modifications to a plan, not for generating the modifications. Koton [1988] has used causal domain models for comprehending diagnostic problems in internal medicine and retrieving appropriate diagnostic cases from memory. Sycara and Navinchandra [1989] similarly have proposed the use of causal domain models for elaborating engineering design problems and retrieving appropriate cases from memory. The KRITIK system uses the model-based approach for three subtasks in case-based reasoning: retrieval of relevant cases from memory, generation of modifications to the retrieved case, and evaluation of the generated modifications. In this paper, we have described only its use of the model-based method for generation of case modifications.

In addition to the differences in tasks and domains, our model-based approach to case-based reasoning differs from that of Simmons and Davis, Koton, and Sycara and Navinchandra, in the representation and organization of the qualitative model it uses. Below, we describe

these differences by comparing our model-based method to Sycara and Navinchandra's. First, while their methods use *causal models*, our method uses deeper *function-structure models*. The states and the state transitions in their models are grounded neither in the function nor in the structure of the system. In contrast, the component-substance model explicitly relates the internal causal behaviors to both the function and the structure of a device, and thus constrains them both from the top and the bottom. Second, the states and the state transitions in their models are represented as character strings. In contrast, the behavioral representation language provides a specific vocabulary for expressing the semantics of behavioral states and state transitions. Third, their methods do not provide any scheme for organizing knowledge of the internal causal behaviors of a system. In contrast, the behavioral representation language provides a specific vocabulary for organizing knowledge of the behaviors around the functions they achieve. Fourth, their models do not provide any typology of structural or behavioral interactions between the components of a system, or of the functional differences and structural differences between cases. In contrast, the component-substance model explicitly provides these typologies. Fifth, as a consequence of above, their model-based methods can handle only very simple forms of causal reasoning. In contrast, our model-based method involves teleological reasoning about the functions of physical devices and topographic reasoning about their structures, in addition to causal reasoning.

As mentioned earlier, the component-substance model and the behavioral representation language integrate and generalize two earlier device representations: the functional representation scheme and the consolidation method. The model and the language are complementary to the commonsense algorithms of Rieger and Grinberg [1978]. The representation of behavioral states and state transitions in our scheme is similar to their representations. The typology of structural, causal, and behavioral relations in our scheme and the types of inferences this scheme enables, complement those used in the commonsense algorithms.

Finally, the model-based method for case adaptation is complementary to Carbonell's [1986] method of derivational analogy. The method of derivational analogy uses knowledge of the problem-solving process by which the agent arrived at the solution to a previous problem. The model-based based, in contrast, uses knowledge of how the solution of the previous problem satisfies its constraints. The interaction between the two approaches is an open research issue.

Acknowledgements This work has benefited from many discussions with B. Chandrasekaran. S. Prabhakar contributed to the programming of the KRITIK system.

References

[Ashley and Rissland 1988] K. Ashley and E. Rissland. A Case-Based Approach to Modeling Legal Expertise. *IEEE Expert*, Summer, 1988.

[Bylander and Chandrasekaran 1985] T. Bylander and B. Chandrasekaran. Understanding Behavior Using Consolidation. *Proc. Ninth International Joint Conference on Artificial Intelligence*, pp. 450-454, 1985, Los Altos, CA: Morgan Kaufmann.

[Carbonell 1986] J. Carbonell. Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition. *Machine Learning: An Artificial Intelligence Approach, Volume II*, R. Michalski, J. Carbonell and T. Mitchell (editors). San Mateo, CA: Morgan Kauffman, 1986.

[Goel 1989] A. Goel. Integration of Case-Based Reasoning and Model-Based Reasoning for Adaptive Design Problem Solving. Ph.D. Dissertation, Dept. of Computer and Information Science, The Ohio State University, Fall 1989.

[Goel and Chandrasekaran 1989] A. Goel and B. Chandrasekaran. Use of Device Models in Design Adaptation. *Proc. of the Second DARPA Case-Based Reasoning Workshop*, pp. 100-109, 1989, Los Altos, CA: Morgan Kaufmann.

[Hammond 1989] K. Hammond. *Case-Based Planning: Viewing Planning as a Memory Task*, 1989, Boston, MA: Academic Press.

[Kolodner and Simpson 1990] J. Kolodner and R. Simpson. The MEDIATOR: Analysis of an Early Case-Based Reasoner. *Cognitive Science*, 13, 1990.

[Koton 1988] P. Koton. Integrating Case-Based and Causal Reasoning. *Proc. Tenth Conference of the Cognitive Science Society*, pp. 167, 1988, Hillsdale, NJ: Lawrence Erlbaum.

[Rieger and Grinberg 1978] C. Rieger and M. Grinberg. A System for Cause-Effect Representation and Simulation for Computer-Aided Design. *Artificial Intelligence and Pattern Recognition in Computer-Aided Design*, J. Latombe (editor), pp. 299-334. Amsterdam, Netherlands: North Holland, 1978.

[Riesbeck and Schank 1989] C. Riesbeck and R. Schank. *Inside Case-Based Reasoning*. Hillsdale, NJ: Erlbaum, 1989.

[Sembugamoorthy and Chandrasekaran 1986] V. Sembugamoorthy and B. Chandrasekaran. Functional Representation of Devices and Compilation of Diagnostic Problem Solving Systems. In *Experience, Memory, and Reasoning*, J. Kolodner and C. Riesbeck (editors), pp. 47-73, Hillsdale, NJ: Erlbaum, 1986.

[Simmons and Davis 1987] R. Simmons and R. Davis. Generate, Test and Debug: Combining Associational Rules and Causal Models. *Proc. Tenth International Joint Conference on Artificial Intelligence*, 1987, Los Altos, CA, Morgan Kaufmann.

[Stallman and Sussman 1977] R. Stallman and G. Sussman. Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis. *Artificial Intelligence*, 9:135-196, 1977.

[Sycara and Navinchandra 1989] K. Sycara and D. Navinchandra. A Process Model of Case-Based Design. *Proc. Eleventh Cognitive Science Society Conference*, 1989, Hillsdale, NJ: Lawrence Erlbaum.