Validated Retrieval in Case–Based Reasoning

Evangelos Simoudis   James Miller

Digital Equipment Corporation
Cambridge Research Lab

CRL 90/2   December 12, 1990

Abstract

We combine simple retrieval with domain-specific validation of retrieved cases to produce a useful practical tool for case-based reasoning. Based on 200 real-world cases, we retrieve between three and six cases over a wide range of new problems. This represents a selectivity ranging from 1.5% to 3%, compared to an average selectivity of only 11% from simple retrieval alone.

1 Introduction

We have combined simple retrieval (based on the similarity of surface features) with domain-specific validation of retrieved cases to produce a useful practical tool for case-based reasoning. Starting with a case base of 200 real-world cases, we have narrowed our consideration to between three and six cases over a wide range of new problems. This represents a selectivity ranging from 1.5% to 3%, compared to an average selectivity of only 11% from this same case base using retrieval without validation. We are applying the same technology to a larger case base in a different domain, and have deployed a related tool with a much larger case base for actual use in the field.

Our work begins with a real-world problem: a computer manufacturer's diagnosis of system software failures. In this domain, diagnostic knowledge exists in several forms: manuals, courses, production rule systems, and knowledge bases. But the predominant starting point in current use is a set of data bases created by recording successfully diagnosed error conditions. In order to diagnose a new failure, non-expert specialists retrieve from a data base previously solved cases that appear superficially similar to the new problem. They then attempt to verify the similarity by performing tests on the new problem and comparing the results with those of each retrieved case. When they become convinced that a previous case is substantially the same as the new problem, they examine the resolution of the old case and report it (possibly amended or edited to more closely fit the new problem) to the customer. Only in rare cases are experts requested to examine problems — most are resolved from the existing data base — and the solutions are then added to the data base.

This existing human system is a conscious use of case-based reasoning (CBR) techniques; we have improved the system by adding to it an automated tool using results from AI case-based reasoning systems. In order to produce a tool of practical value we were forced to examine more closely the task of retrieval in case-based reasoning. Based on our experience we propose an extension to current systems, validated retrieval, that dramatically reduces the number of cases presented to the reasoning component (human or automated) of a case-based system. Validated retrieval relies on domain-specific knowledge about tests used to compare cases retrieved from the case base with newly presented problem cases. Knowledge about the relationships among the various tests is captured in a validation model which we implement as a semantic network[8]. In order to build our validation model we are faced with a classic knowledge acquisition task. By perusing existing data bases used by specialists we are able to acquire this knowledge with a reasonable amount of effort — and with only a small investment of specialists' time.

2 Retrieval in CBR

CBR systems first retrieve a set of cases from a case base and then reason from them to find a solution to a newly posed problem. Existing systems ([1], [2], [3], [4], [9] and [10]) make two assumptions about the initial retrieval of cases from the case base:

1. Very few cases will be retrieved from the case library.
2. The retrieved cases are relevant to the problem being solved.

In many practical applications, retrieval alone is sufficient to solve the difficult part of a task. For example, in our domain of diagnosis of computer software failures, specialists can easily respond to customer problems if they can quickly locate a few similar cases from their collective past experience. For this reason, we have concentrated on the retrieval aspect of case-based reasoning. In MBRTALK[10], also, the essential task is retrieval; the "reasoning" component consists of merely passing the retrieved information directly to an output unit.

2.1 Related Work

Closest to our own work is the work of Koton on CASEY[5], a CBR system which has been applied in the domain of medical diagnosis. CASEY has a justification component whose goal is to determine whether the causal explanation of a retrieved case applies to a new problem. This frequently allows CASEY to avoid invoking its causal model when creating an explanation for a new case. CASEY's justification phase is similar to our validation phase. But there is an important difference between these two systems arising from different assumptions about tests. CASEY relies on precisely two tests (EKG and X-rays), both of which are inexpensive and non-invasive. Both of these tests are performed prior to the retrieval phase and the results are used to provide surface features for the retrieval algorithm. By contrast, there are literally hundreds of tests to be performed in our domains and it is far too expensive to perform all of them in advance of initial case retrieval. As a result, our systems devote attention to minimizing the number of tests that are performed. We not only perform tests incrementally and cache the results, but also employ knowledge about the tests themselves to reduce the number of tests performed.

The CBR system CHEF[2], whose domain is Chinese cooking, also has a justification component. In order to justify each retrieved case, CHEF uses backward chaining rules. While our validation model is not appropriate to this domain, we have examined and rejected the use of rules for our validation models. This decision is based on the difficulty of acquiring the expert knowledge needed to create large rule sets, especially in comparison to the simplicity of constructing our validation models. These models are captured by a semantic network that represents groups of tests and information about the relationships between the groups. Furthermore, as with CASEY, CHEF does not use the results of comparisons with earlier cases to prune its search for relevant cases.

The SWALE[6] system concentrates primarily on modifying the explanation (contained in an explanation packet or XP) of a retrieved case to match a new problem. Nonetheless, it has a subcomponent, XP ACCEPTER, that justifies the application of a retrieved XP to a current situation. The ACCEPTER verifies an XP by determining if it can believe the applicability checks that are packaged with each XP. Each such test is similar to the tests associated with our validation model. Because of the small number of cases (eight, by our count) XP ACCEPTER never addressed the issues of scale which are our major concern. Thus, SWALE never developed a justification model to relate the various applicability checks to one another.

2.2 Two Phases: Retrieval and Validation

Our goal is to take a sizable pre-existing case base along with a new problem and produce a small number of relevant cases. Like our human specialists, our systems perform diagnosis in two phases:

Retrieval: it poses a query to the case base using a subset of the features that describe the new problem.

Validation: it follows the validation procedure from each retrieved case to determine if it applies to the problem at hand.

The goal of the retrieval phase is to extract from the case base those cases that appear to be relevant to the new case. Since the case base is large, and we have been interested primarily in sequential implementations, it is important that the case base be organized in a way that permits efficient search based on surface similarities. For this reason, we organize the cases into a generalization hierarchy (using UNIMEM[7]). The retrieval phase consists of traversing the generalization hierarchy to find a close match to the new problem. The result of this traversal is either an individual case (a leaf node) or a set of cases (an internal node in the hierarchy, returned as all of the cases indexed under that node). Unlike those systems that rely exclusively on UNIMEM for case retrieval, we don't fine tune UNIMEM to reduce the number of cases retrieved.

The validation phase then considers each of the retrieved cases and attempts to show that the case is relevant to the problem at hand. Associated with each case in the case base is a set of tests and their result values that must be met for the stored case to be valid. We call the set of tests and values a **validation procedure**, and each element of this set (i.e. a single test/value pair) is called a **validation step**. The tests are applied to the actual problem and the results are compared with the results in the case. Based on this comparison both the current case and other retrieved cases can be removed from further consideration. Only when all of the tests for a given case are successfully matched against the current problem is the case reported as a candidate for a reasoning component's consideration. (In other domains, it may be possible to assign weights to individual test results and use a threshold or averaging scheme for deciding whether or not to reject the case.)

3 The Validation Model

The validation phase of our method is straightforward if the individual validation tests are simple and self-contained. Unfortunately, in our domains, and probably in most real-world domains, this is not the case. In each domain we have studied, we have found that the tests are interrelated in a way that is not evident in advance, and we have been forced to face the knowledge acquisition task head-on. In Section 3.2 we describe a methodology for acquiring this knowledge about tests. We have successfully used this methodology to develop validation models: structures that capture much of an expert's knowledge in a way that makes it easy for the validation phase to process the tests it requires.

3.1 What is a validation model?

Rather than require a complete and accurate description of each test used by the specialist, we capture the overall structure of the test space itself. The resulting structure, our validation model, consists of related groups of tests and information about the relationship between the groups. For example, if we want to know why a house is hot (the problem), we may first want to see if the air conditioner is working; but this requires us to find out if the house *has* an air conditioner. In this example, the desired test (is the air conditioner working?) is related to another test (is there an air conditioner?). This knowledge, as well as knowledge about the important outcomes and implications of a test, is captured by the validation model. We have chosen to represent this knowledge in the form of a semantic network whose nodes correspond to sets of tests and whose arcs indicate relationships between these sets.

3.2 Creating a Validation Model

We build our validation models by first examining *existing* data bases that are used by human specialists. These data bases may be either formalized (as in the case of our WPS-PLUS[1] system) or merely informal notes prepared by the specialists for their own perusal (as in the case of our VAX/VMS system). In our two case-based systems, the existing data contains a textual description of the steps that the specialists used to verify a hypothetical explanation of the problem. In constructing the validation model, it is our goal to capture the interrelationships between the validation tests. As a result, we have built validation models that correspond to a particular case base by:

1. Reading the validation procedures of each case and building a list of all the validation steps used in the entire data base. In the process of reading the data base and preparing this list, the implementor develops a sense of the underlying (but unstated) relationships between tests that are mentioned in the data base.

[1]DEC, VAX, VMS and WPS-PLUS are registered trademarks of Digital Equipment Corporation.

2. Examining the resulting list, looking for groups of tests that appear to form related sets. Organizing the list provides a basis for discussion with domain experts, who help "debug" the proposed organization.

3. Refining the structure of the list through knowledge acquisition sessions with domain experts. During these sessions, significant ranges of test results are identified, as are inferences from these results that eliminate the need to perform other tests. That is, a dependency graph based on test results is developed.

4. Iterating the above two steps after consulting additional information such as manuals and code documentation. The structure of the domain becomes clearer at each iteration. (We have found that three iterations are sufficient to produce a useful structuring.) The final validation model consists primarily of entries corresponding directly to information that appears in the original data base.

5. Integrating the test sets into the structure derived in the previous step. This integration makes explicit the prerequisites of each test, as well as providing alternative ways of obtaining information ordinarily provided by a particular critical test in cases where that test cannot be performed.

4 An Extended Example

In order to understand the validated retrieval process, consider the following example. Our domain is automobile diagnosis and repair, and we assume an existing case base with its associated validation model. We are given the following case:

NEW CASE					
make:	MAZDA	**model**:	626	**model year**:	1985
engine type:	2.0L EFI	**miles**:	50,000		
problem:	engine does not start.				

The retrieval phase uses the make, model, problem, and approximate year of manufacture to search through a case base of previous automobile problems. Based on these surface features, we retrieve three cases to be validated before presentation to a reasoning component:

CASE 1					
make:	MAZDA	**model**:	626	**model year**:	1988
engine type:	2.0L EFI	**miles**:	10,000		
problem:	engine does not start.				
validation:	The fuel injector was clogged. Fuel was not delivered to the combustion chamber for the engine to ignite. For this reason the engine could not start.				
solution:	cleaned the fuel injector.				
CASE 2					
make:	MAZDA	**model**:	626	**model year**:	1984
engine type:	2.0L	**miles**:	60,000		
problem:	engine does not start.				
validation:	The car had a faulty gas pump. Fuel could not be delivered to the combustion chamber. For this reason the engine could not start.				
solution:	Replaced the gas pump.				
CASE 3					
make:	MAZDA	**model**:	626	**model year**:	1987
engine type:	1.8L	**miles**:	20,000		
problem:	engine does not start.				
validation:	A leak existed in the gas line. Fuel could not be delivered through the fuel line. For this reason the engine could not start.				
solution:	Fixed the leak.				

The validation model contains (at least) the three tests that are referenced by these cases: "check if a fuel injector is clogged", "check if the gas pump is working", and "check if there is a leak in the fuel line". The first of these is actually composed of two simpler tests: a test for fuel present in the reservoir of the injector and a test for fuel exiting the injector's nozzle. If there is no fuel in the injector then we can deduce that the injector is *not* at fault. Rather, the problem lies earlier in the fuel system — either in the pump or the fuel line.

The system first attempts to validate Case 1 by repeating the validation steps from that case. That is, we wish to test if the fuel injector is clogged. In the process of performing this two-step test we actually acquire knowledge that is relevant to Cases 2 and 3: if the fuel reservoir is not empty we can eliminate both cases; if it *is* empty, we can eliminate Case 1. This relationship is encoded in the semantic network that represents our validation model and is used in the validation phase.

In the best case, this validation model allows us to reduce the work required to validate cases from four tests to two tests and simultaneously reduces the number of cases to be considered by the reasoner from three to one (selectivity of 33.3%). The first test is for an empty fuel reservoir; if the reservoir is full then Cases 2 and 3 are eliminated. We then test the nozzle for fuel exiting. If no fuel leaves the nozzle, then Case 1 is presented to the reasoner; but if fuel *is* leaving the nozzle we, unfortunately, eliminate Case 1 as well and leave the reasoning component to its own resources. The worst case requires all four tests and provides either zero or one case to the reasoner.

5 Recent Results

5.1 An Operating System: VMS

The first system we developed is used for the diagnosis of device driver induced crashes of Digital's VMS operating system. The knowledge about surface features was obtained primarily from DEC internal publications and was complemented by an expert from the VMS support team during three knowledge acquisition sessions. It took a total of 84 hours to acquire the domain specific knowledge about surface features. Based on this information, the domain knowledge used by UNIMEM in order to organize the cases into a generalization hierarchy was implemented in five days.

It took an additional four days of reading validation procedures in the data base to develop a validation model for device drivers. In addition, four more knowledge acquisition sessions, lasting 40 hours, were needed to refine and improve the validation model. Encoding the actual validation model took about 80 additional days. The total number of days spent on knowledge acquisition and development is shown below:

Activity	*Person Days*
Knowledge acquisition	20
Development	85

Since this was our first attempt to build a case base and validation model, these numbers are much larger than we expect for subsequent systems. Our work to date on the system described in Section 5.2 appears to confirm this expectation.

The system was evaluated using a case base of 200 cases that were obtained from notes written by specialists. The surface feature retrieval phase of the system was evaluated by presenting each of the 200 cases to the retriever (as new problems) and preparing a histogram of the number of cases retrieved. UNIMEM provides a mechanism, known as **retrieval weights**, for tuning its retrieval capabilities. After some experimentation, we discovered that the use of larger retrieval weights (i.e. more stringent matching criteria) caused the retriever to miss many relevant cases and, in many occasions, to fail to retrieve any cases at all. With less stringent criteria this problem was rectified. However, many of the retrieved cases were not relevant to the problem. With the optimal weighting, we were able to retrieve on average 22 cases per retrieval (11%). The validation phase, however, was able to reduce this number of cases to an average of 4.5 cases out of 200 (2.25%).

In addition, we presented three new cases to the system. Based on surface features alone, we retrieved 20, 25, and 16 cases (10, 12.5, and 8% selectivity). The validation phase reduced this to 3, 5 and 3 cases, respectively (1.5, 2.5, and 1.5% selectivity). Our experts confirm that these validated cases are the only ones relevant to the problems presented.

5.2 A Word Processing System: WPS-PLUS

The second system performs diagnosis of customer problems with the word processing component of an office automation product. During 15 hours of knowledge acquisition sessions, the knowledge about surface features was obtained from a support engineer for the product. It then took an additional five days to encode the domain knowledge for use by UNIMEM.

The validation model was obtained from the validation procedures of the cases in the data base, an internal publication, and 10 hours of knowledge acquisition with the same engineer. While the work is not yet complete (only 50 out of 340 cases have been encoded), it has taken only 10 days to implement the validation model.

This system is still under development. However, the time we spent on knowledge acquisition and development is shown below:

Activity	*Person Days*
Knowledge acquisition	5
Development	10

This system was evaluated using a case base of 340 cases. Repeating the same experiment performed with the VMS case base led to an average of 26 cases per retrieval, or 7.6% selectivity. The validation phase reduced this to two cases, or 0.58% selectivity. Since the validation model for this case base is not yet fully encoded, we have not presented new problems to the system.

6 Scope of Work

Our validated retrieval method can be applied in many types of tasks. The basic requirements are: an existing data base of previous practical experience; a set of quick tests that serve to reduce the search space at low cost; a set of more expensive tests that can further reduce the search space; and an understanding of the relationships between the expensive tests.

We have identified four areas of potential interest, but we have limited our implementation work to the first of these:

- **Diagnostic tasks**. As shown in the example, we use the symptoms of a problem as the surface features for the retrieval phase. The validation procedure describes which tests to perform in order to determine if the case is relevant to the new problem.

- **Design tasks**. The surface features are specifications that a design must satisfy. The validation procedures verify that a proposed design meets the specification.

- **Sales tasks**. The technique can be used to help identify sales prospects for a new product. The surface features are the characteristics of a customer such as: size of business, type of business, location of business. Each validation procedure describes the customer's requirements that were satisfied in a previous sale. The validation model includes tests that determine whether or not a customer needs a particular type of product.

- **Management tasks**. These tasks include accounting, credit analysis, investment decisions, and insurance underwriting. In each of these areas, specialists can identify easily recognized features in their problem domain (type of company, size, etc.) that allow rapid retrieval of similar situations encountered in the past. They then have more detailed tests that can be applied (debt/equity ratio, payment history, type of client, balance sheets, etc.).

7 Conclusions

Our work has concentrated exclusively on the issue of case retrieval. A careful study of two applications in which people consciously use case retrieval has shown that retrieval based solely on surface features is not sufficiently discriminating for use with large case bases. It results in large numbers of cases returned to the reasoner, each of which must then be further examined at great expense. Adding a validation phase that uses knowledge of domain-specific tests to prune the retrieved cases dramatically reduces the number of cases that must be examined by the reasoner.

We have found that acquiring knowledge about domain-specific tests is aided by an initial perusal of the existing data base used by specialists. With a reasonable amount of effort, and with only a small investment of specialists' time, this information can be captured in a validation model represented as a semantic network. We have used this methodology to produce two systems. One of these systems has been successful in practice, and the other (incomplete) system is likely to be equally useful.

While the burden of knowledge acquisition in our methodology is small compared with other methods, it is not negligible. Automating this work by combining a natural language system to analyze existing data bases with AI-assisted statistical comparison of surface features provides a fertile area for further investigation. Furthermore, we suspect that a careful study of such a system in practice will reveal validation tests that are sufficiently common that it may be reasonable to promote them to surface features, leading to a system with better retrieval capabilities. This analysis, which must first be performed manually to validate our assumption, is itself an excellent area for the application AI methods.

Acknowledgements

The authors wish to thank Prof. David Waltz for his help with this research. In addition, we have received helpful comments from Mark Adler, Andrew Black, Dave Hanssen, Rose Horner, and Candy Sidner. We are also indebted to the engineers who have given their time and knowledge to help us understand the two domains we have studied: the Digital Support Engineers at Colorado Springs, Colorado and Spitbrook, New Hampshire.

References

[1] Ray Bareiss, Karl Branting, and Bruce Porter. The role of explanation in exemplar-based classification and learning. In *Proceedings of Case-Based Reasoning Workshop*, 1988.

[2] Kristian Hammond. *Case-Based Planning: An Integrated Theory of Planning, Learning, and Memory.* PhD thesis, Yale University, 1986.

[3] Janet L. Kolodner. Reconstructive memory: A computer model. *Cognitive Science Journal*, 7:281–328, 1983.

[4] Janet L. Kolodner, Jr. Robert L. Simpson, and Katia Sycara-Cyranski. A process model of case-based reasoning in problem solving. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1985.

[5] Phyllis Koton. *Using experience in learning and problem solving.* PhD thesis, Massachussetts Institute of Technology, 1988.

[6] David Leake. Evaluating explanations. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, 1988.

[7] Michael Lebowitz. Experiments with incremental concept formation: Unimem. *Machine Learning*, 2:103–138, 1987.

[8] M.R. Quillian. Semantic memory. In Marvin Minsky, editor, *Semantic Information Processing*, pages 227–270. MIT Press, 1968.

[9] Edwina Rissland and Kenneth Ashley. Hypotheticals as heuristic device. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, 1986.

[10] Craig Stanfill and David Waltz. Toward memory-based reasoning. *CACM*, 29:1213–1228, 1986.

A Creating the VMS Validation Model

This appendix describes, in detail, the method used to construct the validation model for the VAX/VMS device driver diagnosis system. Each subsection corresponds to one of the steps described in Section 3.2. For expository purposes, of course, we have removed a number of intermediate steps and show only selected results.

A.1 Reading the Initial Data Base

We begin with the data base in current use by Customer Support Specialists. This data base (actually, an informal textual "notes file") consists of about 150 entries. A typical entry is shown in Figure 1. At this stage we not only read these entries, but we tidy them up a bit. For example, some of these entries were incomplete and were manually expanded into multiple cases for our case base, resulting in 200 entries to be studied.

The primary purpose of this initial reading is to create a list of all of the tests that are explicitly mentioned by the specialists in their explanations of the resolved cases. From the case presented in Figure 1 we extract three tests:

1. Retrieve Process PCB
2. Check JIB Address
3. Check Count

For this particular data base, we derived an initial set of about 100 tests.

A.2 Grouping the Tests

Reading through this data base led to a natural decomposition of the tests by grouping them according to data structures mentioned in the cases. Thus, we grouped tests related to the JIB (job information block) together, joining the tests "Check JIB Address" and "Check Count" from the case shown in Figure 1. In this particular data base, we grouped the tests into roughly seven sets of tests.

By contrast, in our work with the WPS-PLUS data base, the natural grouping was along functional lines. While we have no firm evidence, it seems likely that such "natural" groupings will occur in most sizable data bases.

Version of system:	VAX/VMS V4.2
Reason for bugcheck exception:	ACCESS VIOLATION
Process currently executing:	

```
SP=>  8018E7AC  00000004
      8018E7B0  7FFE7DE4
      8018E7B4  FFFFFFFD
      8018E7B8  0000026A
      8018E7BC  00000000  !THIS SHOULD HAVE BEEN ON
                          PREVIOUS LINE
                          !AND VISA VERSA
      8018E7C0  00000001
      8018E7C4  00000005  !BEGINNING OF SIGNAL ARRAY
      8018E7C8  0000000C  !ACCES VIOLATION
      8018E7CC  00000004  !WRITE ACCESS PROTECTION
      8018E7D0  00000020  !VA
      8018E7D4  80004A4E  IOC$BUFPOST+017
      8018E7D8  04040000
      8018E7DC  80004A6D  IOC$BUFPOST+036
      8018E7E0  800CD108  DZDRIVER+118
```

the reason for this bugcheck is because the pid field of the IRP has been zeroed. IN post processing the system tries to give the buffered byte count back to the process that did the io. HE does this by using the PID field of the irp to index into the pid table then gets the PCB(null process in this case) and gets the JIB address. THE JIB address for the null process is 0. THE system then goes to the JIB$L_BYTCNT offset (20) from the base of the JIB (0 in this case) and tries to add back in the byte count. IN this case we get an access violation.

have response from VMS developement that there is/was a bug fixed in 4.2 it has been mentioned to try and lower the baud rate on terminals as a workaround.

Figure 1: An Initial Data Base Entry

A.3 Refining the Test-Group Structure

Armed with this group of tests, we returned to the specialists who had prepared the data base. With their help, we learned the meaning of each of the tests we had identified. The specialists pointed out that our natural groupings didn't correspond to groupings that they would have made — primarily because they had a (non-articulated) set of abstractions that our groupings did not make clear. By reviewing our groupings they were forced to verbalize the abstractions that they took for granted, and we were able to sort our tests into 15 groupings that reflected the specialists' notion of the correct abstractions for their domain.

In the case of the VAX/VMS data base, the specialists typically subdivided our groupings into three distinct categories: those that test for the existence of the expected location in memory, those that tested the internal consistency of the data structure, and those that probe the data structures for specific value. In the case of the tests extracted from Figure 1, "Check JIB Address" is in the first category, the specialists pointed out for the need for an additional test we had not identified that performed the consistency test, and "Check Count" is in the third category.

Thus, our initial model of seven structure-based test groups was refined, through interaction with specialists, into 15 groups with a layered structure. Testing always proceeds from one layer to another, starting with tests for existence of the data structure, proceeding to tests on the consistency of the data structure, and finally on to probing specific values within the data structure.

A.4 Acquiring Additional Knowledge

This step is primarily an iteration of the earlier one, but two unusual items arose during this phase for the VAX/VMS system. The first was the realization that certain terms used in the descriptions were not reflected in any of the tests our groupings we had already derived. In this particular case, there was continual use of phrases like "during AST delivery," and "during post-processing." Examination of additional material (material from an internal DEC course suggested by the specialists as a good starting point) revealed that these reflected the processing stages of the system we were diagnosing. Our initial breakdown had been along data structure boundaries, but an orthogonal, equally important structuring is possible along temporal lines based on these stages. It is these two alternative decompositions that we referred to in Section 3.2 as the "structure of the domain."

Further iteration with the course materials and the specialists reveals a much richer structure, consisting of multiple levels of abstraction. With the temporal decomposition taken as primary, Figures 2 and 3 show two different levels of abstraction. Figure 2 is taken at a very abstract level, showing the four major processing stages and the data structures used at each stage. Figure 3 is a "close-up" of the single processing stage known as "AST delivery."

A.5 Integrating Tests into the Model

Armed with a much-improved model of the domain, we built a semantic network capturing as much of this model as bears directly on the ability to select and apply the tests and test groups. In the VAX/VMS system, our validation model reflects both decompositions along data structures and along processing stages. The semantic network has 15 nodes, for each test group, 77 nodes that represent knowledge about the processing stages of device drivers, and 22 nodes that represent knowledge about data structures.

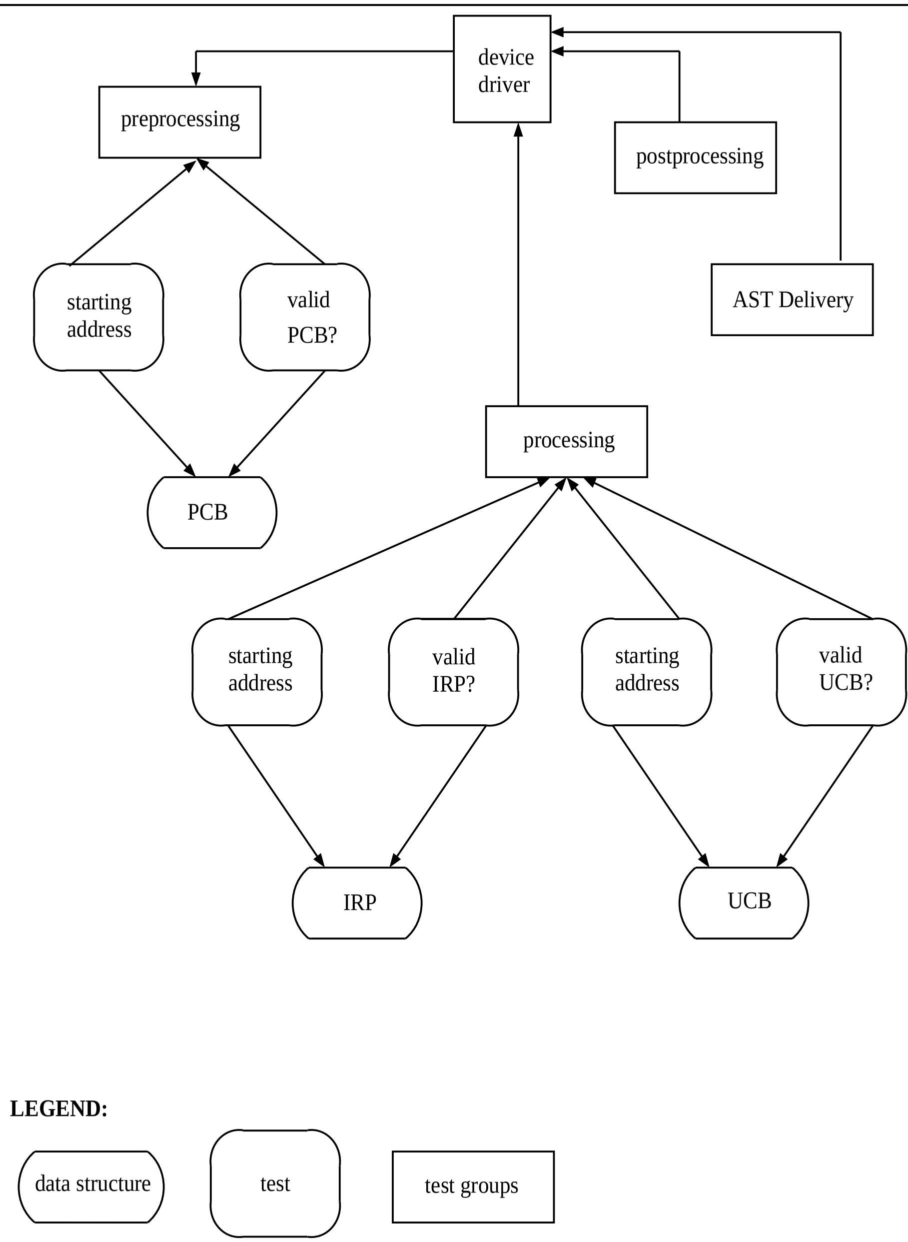

Figure 2: Device Driver Validation Model: Abstract View

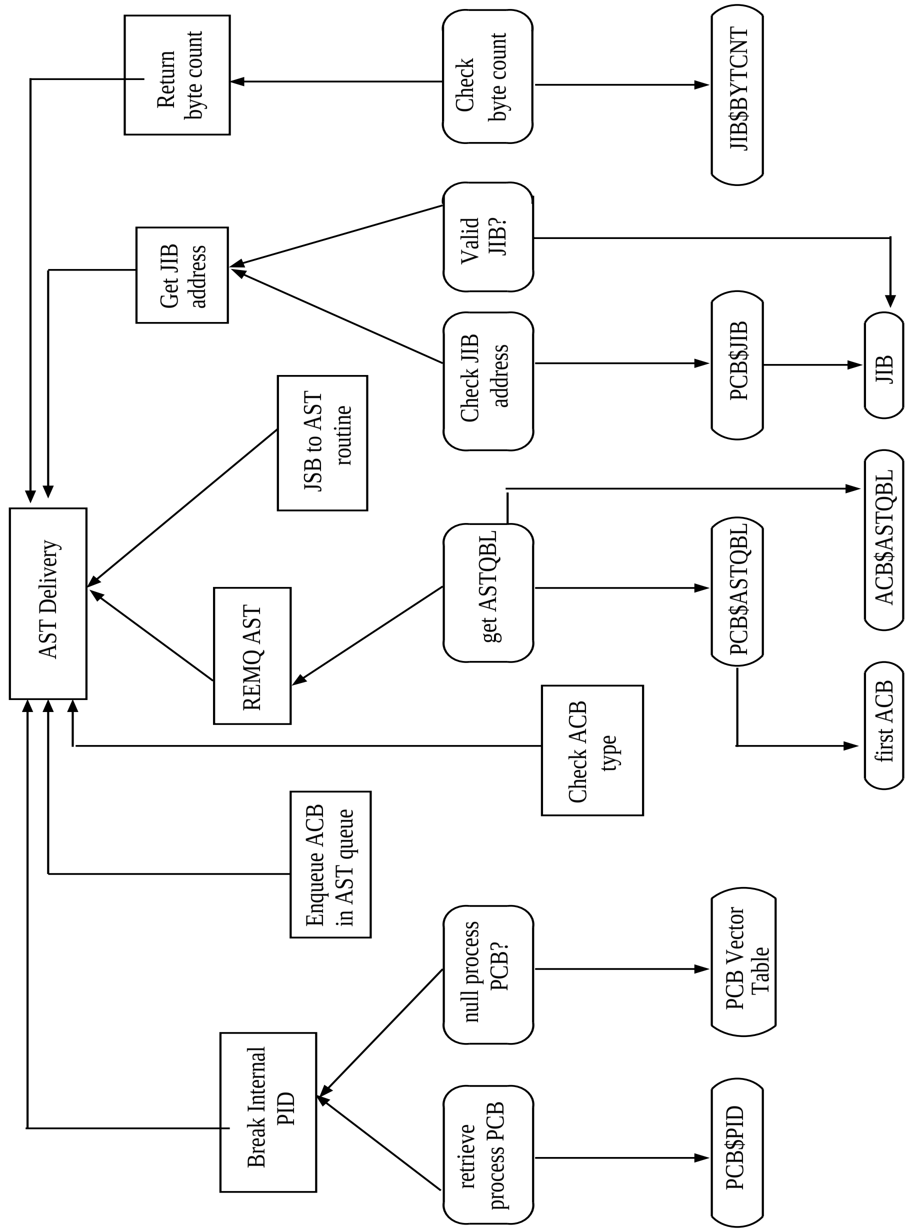

Figure 3: Device Driver Validation Model: AST Delivery