Performance Assessment and Incremental Evaluation of Rule-based Systems

P. G. Chander, R. Shinghal, and T. Radhakrishnan
Department of Computer Science,
Concordia University,
Montreal, Canada H3G 1M8
{gokul, shinghal, krishnan}@cs.concordia.ca

Abstract

Rule-based systems that are easily testable are required for high reliability applications. However, as a rule base evolves, developers prefer incremental evaluation owing to the high cost of regression testing. For quality and reliability improvement, researchers advocate that the evaluation phase be integrated with development: thus, incremental evaluation becomes more important in this context. In this paper, we propose a three-tiered life-cycle model for integrating evaluation in a rule-based system life cycle. We then explore two specific issues from a system's evaluation perspective: (1) the limitation of functional testing for performance evaluation; and (2) the optimization of the cost and effort for evaluation by using incremental methods, whenever possible. This research was motivated by our experience in the design and evaluation of rule-based systems.

Introduction and Motivation

The design of rule bases is often plagued by various anomalies whose detection requires a variety of evaluation procedures (O'Keefe & Lee 1990; O'Keefe & O'Leary 1993; Kiper 1992; Guida & Mauri 1993). The evaluation processes have been traditionally classified as verification, validation, and performance analysis (or measurement). Verification detects the anomalies due to redundancy, deficiency, circularity and ambivalence; functional validation tests the conformance of a system with its functional requirements; structural validation checks whether the observed functional performance is the result of the correct structure (rule) interactions and tries to capture the system structure in a well defined way; and performance assessment measures the adequacy, optimality, and test case coverage of the system. Not only are these processes non-trivial, but system developers are often unclear where they fit in a system life-cycle (Hamilton, Kelley, & Culbert 1991).

One of the primary reasons that makes the evaluation of rule-based systems harder is the lack of a well defined link between system conception and its realization. In fact, in many cases, there does not even seem to be a design level to weigh the various compromises with which domain knowledge and constraints can be represented in the rule base. For rule base designs that integrate verification and validation (V&V) processes with ease, we need a well defined link that connects the following three stages in its development: (1) the functional requirements of the system, (2) the rule base design constraints, and (3) the implementation of the rule base (which, in turn, determines the extent of evaluation that can be performed on the system). This is portrayed in Figure 1. It shows a development perspective for rule-based systems.

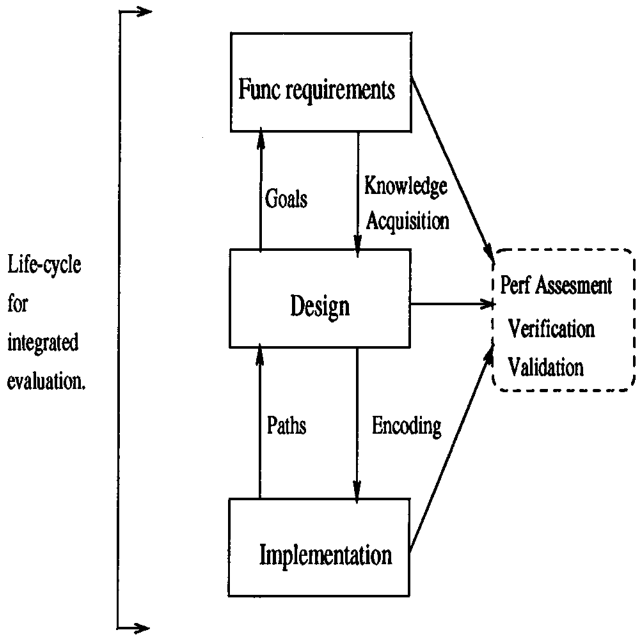

Figure 1: A simple software development perspective for rule-based systems for viewing evaluation as part of all the phases of a system's life-cycle.

Our motivation arises partly from our experience in the design and evaluation of rule-based systems, and from the increasing emphasis for integrating evaluation and reducing evaluation costs (Meseguer

1992; Lee & O'Keefe 1994; Chander 1996). Contemporary researchers believe that quality and reliability improvement for rule-based systems can be obtained through formal approaches to system construction and integrated evaluation (Plant 1992; Krause *et al.* 1993; Lee & O'Keefe 1994). Integrating evaluation in a system's life-cycle, however, is non-trivial as costs can be prohibitive if all tests are (automatically) repeated for every modification to the system. It has also been pointed out that functional or random testing approaches alone do not test a system sufficiently (Plant 1992). Based upon these observations, and from our experience, we believe that integrating evaluation in a system's life-cycle should take into account the following two fundamental aspects: (1) the role of a system's structure for its evaluation; and (2) the methods for incrementally evaluating a system to reduce evaluation costs.

The paper is organized as follows. Section 2 describes a knowledge representation scheme that uses rule sequences and a domain knowledge reference obtained during knowledge acquisition (Grossner *et al.* 1996). Section 3 describes a performance evaluation mechanism and outlines the limitation of the commonly used functional testing approach in this case. Section 4 deals with the the problem of incremental evaluation. Concluding remarks appear in section 5. Details of the design phase (see Figure 1) are given in (Chander, Radhakrishnan, & Shinghal 1997) and are not discussed here.

Knowledge Representation Using Paths

Formal and rigorous approaches to knowledge acquisition for rule-based system development are favored by researchers (Krause *et al.* 1993). Our model of a rule-based system is based on viewing problem solving as goal-to-goal progressions (Shinghal 1992), and in modeling these goal-to-goal progressions in a rule base.

In this approach, the domain expert specifies a set of significant states that need to be reached to solve problems in the domain. For example, in a medical diagnosis domain, inferring a `liver-disease` is a significant state toward inferring `liver-cirrhosis` as a final diagnosis. Typically, the domain expert specifies concepts associated with the domain that serve as significant states for problem solving, and the knowledge engineer translates these concepts into a representation language, say a conjunction of first-order logic atoms that capture the intent of the domain expert. Such states are called *goals*. In addition, the domain expert also specifies *inviolables*, which are constraints associated with the domain; an inviolable is a conjunction of atoms that should never be true, for example, $MALE(x) \wedge PREGNANT(x)$; obviously, no goal should contain an inviolable.

Definition 1 (Goal Specification, Goal and Non-goal Atoms) *The set of goals and inviolables of a domain constitutes the goal specification of that domain. The atoms that are goal constituents are called goal-atoms; the other atoms are non-goal, they being needed for rule encoding.*

A rule base constructed based on a given goal specification, implements problem solving by rule sequences that progress from goal to goal. The complete knowledge of the domain is represented as goals and the rule sequences that infer these goals causing a progression in problem solving. This progression can conceptually be portrayed as traversing an AND/OR graph called the **goal graph of the domain**, or, simply, the goal graph (see Figure 2). Each node in a goal graph corresponds to a goal, where the unshaded nodes denote goals that are not solutions, and the shaded nodes denote solutions. A **connector** in this graph is from a set of goals $G = \{g_{i_1}, g_{i_2}, \ldots g_{i_n}\}$ (the children) to a goal g (the parent), where $g \notin G$. A circular mark in the connector near g indicates that goal g is inferable from goal g_{i_1} and goal g_{i_2} and ... goal g_{i_n}. The different connectors to a goal depict the different alternatives for inferring that goal. Permissible combination of initial evidence are said to be level-0 goals; other goals are at higher levels as portrayed by the first digit in their subscript.

A goal graph is a conceptual portrayal of problem solving at the *functional level.* At the implementation level, a connector from G to g represents a set of rule sequences that must be fired to infer g from G. If a rule r_j occurs immediately after rule r_i in such a sequence, then a non-goal atom in the consequent of r_i unifies with an atom in the antecedent of r_j; goal atoms inferred within the rule sequence contribute only to the goal being inferred. These partially ordered rule sequences are called **paths** (Kiper 1992; Grossner *et al.* 1996).

The extent to which a given rule-based system realizes the acquired knowledge of goal inference is reflected by the paths in its rule base; they are collectively said to portray the **structure** of the rule base (Chang, Combs, & Stachowitz 1990; Kiper 1992; Grossner *et al.* 1996).

Definition 2 (Rule Base Structure) *The structure of a rule base (or, simply structure) is defined as* $< \mathcal{G}, \Sigma >$ *where* $\mathcal{G}$ *is the goal specification of the domain, and* Σ *is a set of paths in the rule base such that,*

$$(\forall \Phi \in \Sigma)(\exists G, g)(G \subset \mathcal{G})(g \in \mathcal{G})\ G \wedge \Phi \vdash g$$

Paths in a rule base have been the basis for a variety of evaluation processes (Chang, Combs, & Stachowitz 1990; Kiper 1992; Grossner *et al.* 1996). However, the relation of the paths to the acquired domain knowledge is implicit. By making this link explicit via the goals (cf. the above definition of

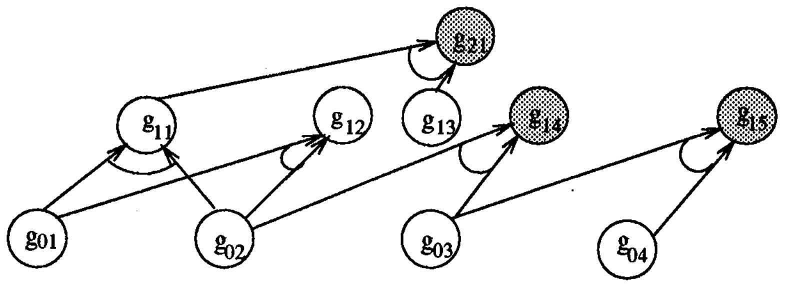

Figure 2: A goal graph of an abstract domain.

$$\begin{aligned}1 &: a \rightarrow b\\ 2 &: b \rightarrow c\\ 3 &: a \rightarrow d \wedge e\\ 4 &: d \rightarrow f\\ 5 &: e \rightarrow g\end{aligned}$$

Initial evidence : a. Goals: c, f, g

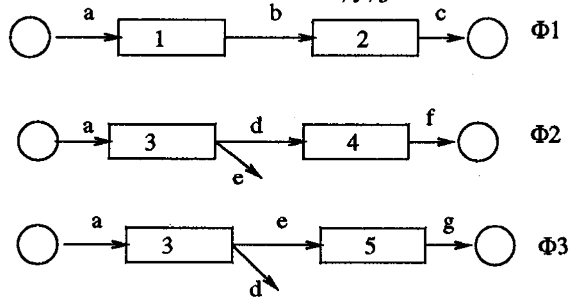

Figure 3: An example of a rule base, goals, and paths.

rule base structure), we capture the rule interactions that pertain to problem solving in a meaningful manner because these rule interactions can be mapped to inferring goals. In addition, rule interactions in a path are completely *localized*: by its definition, all rules in a path are ready to fire once the goals required by the path are inferred. A software tool called Path Hunter is available to extract paths from a rule base (Grossner *et al.* 1996). Figure 3 show a sample rule base, goals and paths.

Limitation of Functional Testing for Performance Assessment

An **optimal** system is one where every inferred goal counts toward a useful unit of work for solving a given problem. In a non-optimal system, however, errors due to knowledge encoding in the rules can result in a large number of rule firings that are not useful. Clear definitions are needed to abstract problem solving in order to measure the extent of redundant work done in the system. Similarly, an **adequate** system is one that can infer a solution for every permissible combination of initial evidence. The extent of the adequacy of a system in its operating domain is an important attribute particularly for safety-critical systems (Giovanni 1989). Below, we describe how goals, viewed as meaningful units of work accomplished, can be used to assess the optimality and adequacy of a system. Test case coverage analysis of a system is not discussed here to conserve space. We refer the reader to Preece *et. al.* (1993).

Given the initial evidence corresponding to a problem, if the system infers some final goals that correspond to the solution of the problem, then the inferred final goals are said to be *relevant*. An intermediate goal is relevant iff there exists a problem in the domain whose solution has this intermediate goal as its ancestor.[1] A final (intermediate) goal is irrelevant iff it is not a relevant final (intermediate) goal. An irrelevant final goal does not constitute a solution to any problem in the domain; it reflects on the capability of the system, that is, its possible inadequacy (Guida, 89). Irrelevant intermediate goals indicate that the system at times may do redundant work by inferring goals that are never used as part of any solution; this may suggest that the system may be functioning sub-optimally in its domain. A system is **goal-relevant** iff all its goals are relevant. A system is said to be **goal-irrelevant** iff it is not goal-relevant. For example, in figure 2 intermediate goal g_{12} and final goal g_{21} are irrelevant.

One way to check whether a system is goal-relevant is to test the system on a set of problems $P_1, P_2, P_3, \ldots$ from the domain. For each P_i we obtain a set γ_i that records a list of intermediate goals that are causal to a final goal (see Figure 4). However, testing to determine a system's optimality and adequacy is limited by the following theorem.

Theorem 1 *The determination of goal-relevancy by testing (that is, using a set of test cases) is semi-decidable.*

Informal Proof Outline. For every P_i used as

[1]The ancestor of a node g in a goal graph is a node g_a that is either its parent, or a parent of one of the ancestors of g.

i_1
i_2
i_3
f_1
f_2
i_4
i_5
f_3

Goals observed in testing the system on problem P_j.

The set γ (partial) from this test is,

$$\gamma_j = \{ \langle\{i_1, i_2, i_3\}, f_1\rangle, \langle\{i_1, i_2, i_3\}, f_2\rangle, \langle\{i_1, i_2, i_3, i_4, i_5\}, f_3\rangle \}$$

This mapping γ_j extracted is partial because the set of goals inferred is specific to problem P_j. In this figure, for clarity, we have used the notation $i_1, i_2, \ldots$ to indicate intermediate goals and $f_1, f_2, \ldots$ to indicate final goals.

Figure 4: Determination of optimality and adequacy by testing.

a test case, we obtain the corresponding set γ_i by the procedure outlined above. If a stage comes that $\gamma_1 \cup \gamma_2 \cup \gamma_3 \cup \ldots$ contain all the goals in the system, then we terminate the procedure since the system is goal-relevant. The disadvantage of this approach is that this procedure may **never** terminate. If after generating γ_i, the procedure has not terminated because the union of the γ's collected so far does not include all the goals, it is not necessarily true that this procedure will terminate after generating γ_{i+1}. The rapidity with which the procedure terminates, if it does at all, depends upon the problems selected from the domain which can be infinite. The procedure is thus semi-decidable in general. ∎

Note that the semi-decidability arises due to the decision of using a functional testing strategy on all problems in the domain to determine these attributes. The problem of optimality and adequacy determination itself is not semi-decidable. However, most developers tend to use a functional testing approach for their systems and often refrain from performing any structure-based testing (Hamilton, Kelley, & Culbert 1991). Thus, the above theorem is important to a developer: it emphasizes the need to determine the performance measures optimality and adequacy through an examination of the system structure by pointing out the limitation of the commonly used functional testing (see also (Plant 1992)).

The determination of relevant and irrelevant goals is, in general, non trivial because a given intermediate goal may have a path inferring that goal, yet it can be irrelevant. This can happen because this goal was never used to infer any other final goal. Similarly, a final goal may be inferred by a path, but the rules in the path may never be enabled because no permissible combination of initial evidence would be causal to do so. Thus, it is necessary to enumerate all *path sequences* from permissible initial evidence to final goals to determine the goal dependencies. In other words, a procedure to extract the goal graph from a rule base, using paths as connectors, is required because we treat permissible combination of initial evidence as level-0 goals.

A goal graph extraction should enumerate all the connector chains from permissible initial evidence to a final goal. This progression can be captured by a sequence of paths required from some given initial evidence to a final goal. Such a sequence of paths is called a **route**.

Definition 3 (Relevant and Irrelevant Routes) *A route is a sequence of one or more paths. A route from a level-0 goal to a solution is called a* **relevant** *route; any other route from a level-n goal to a goal g', where $n \geq 1$ and g' is not a solution, is called an* **irrelevant** *route.*

Clearly, every irrelevant route is indicative of goal irrelevancy in the system. By recording whether an irrelevant route terminates on an intermediate goal or a final goal, and the number of such irrelevant routes, the extent of sub-optimality and inadequacy of the system can be quantified. An algorithm to determine these attributes is given in (Chander 1996). An informal description appears in Figure 5.

Incremental Evaluation to Reduce Costs

Rule bases evolve by test results, by feedback of users on test operation, and by the changing requirements of a domain. Every evolution should, of course, be tested to confirm its correct operation with respect to the change(s) in the system. For example, adding one more rule could result in changed paths, thus altering the goal relevancy status of the system. Hence, all the paths should be extracted to determine the effect of the change. However, extracting all the paths once again even for a small change (that is, a regression testing approach) can entail a large cost (Meseguer 1992) as path extraction has an exponential complexity in the worst case (Grossner *et al.* 1996). In this section, we state the problem of incremental evaluation, and outline a method for incremental path extraction.

More generally, incremental evaluation can be

```
Obtaining all relevant routes, and
irrelevant routes culminating on a final
goal.
1. Select a final goal $f$.
2. Starting from $f$, build recursively the
   goal graph containing all the goals from
   which $f$ can be inferred.
Obtaining irrelevant routes that terminate
on intermediate goals.
1. Start from every intermediate goal $i$
   that has not appeared in any of the
   routes after steps 1 and 2 of the above
   procedure are applied to all the final
   goals.
2. For such intermediate goals $i$ build
   recursively the goal graph containing
   all the goals from which $i$ can be
   inferred to obtain irrelevant routes
   that culminate on the intermediate goal
   $i$.
```

Figure 5: Determination of relevant and irrelevant routes in a goal graph.

viewed as an optimization problem, where we wish to minimize the cost of evaluation while evaluating the system due to some modifications without compromising on the quality and reliability of test results.

Problem Statement 1
(Incremental Evaluation) *Let S and R be the system specification and rule base respectively. Let the operator Δ denote a small change of an entity. Let C be the cost of testing the system so that ΔS and ΔR represent changes in system specification and rule base modifications, respectively. Determine an evaluation strategy that is based upon ΔR and ΔS so that the cost for evaluation is comparable to that change (that is, it should be ΔC, as opposed to C). For example, if the change to a rule base is 10% (with respect to some reference), then the cost of evaluation should be of the order to 10% of C so that the total cost for testing taking the modifications into account should be $1.1C$ ($C + 0.1C$) as opposed to $2C$ (repeating the tests).*

In our framework, all our system evaluation perspectives (verification, validation, and performance and quality assessment) are based upon paths and goals (Chander 1996). Hence, the above problem transforms to incremental path extraction: that is, identify and extract only those paths that are affected by a change in the rule base. In the description below, we assume that the goal specification has not changed, and modifications involve only non-goal atoms. The procedure taking into account changed goal specification as well as rule base modifications is complicated, and is currently under our investigation.

Given the changes to a rule base ΔR, to identify paths that require re-extraction, we must first identify paths that are potentially affected by that change $\Delta\Phi$, where Φ is the set of paths in a rule base. From $\Delta\Phi$, we then obtain $\Delta\Phi'$, the set of paths that require re-extraction. In the worst case, (for regression testing) $\Delta\Phi' \equiv \Delta\Phi \equiv \Phi$. Let $R = \{r_1, r_2, \ldots r_n\}$ and $\Delta R = \{r'_1, r'_2, \ldots r'_m\}$, $0 \leq |\Delta R| \leq |R|$, where we use the notation $|S|$ to denote the size of a set S.

Since the goal specification is unchanged, the changes to a rule cannot affect existing goals; thus, $\Delta\Phi$ can be easily computed ($O(|\Phi| * |\Delta R|)$) by noting every path that contain a rule from ΔR. The path requires re-extraction, iff the rule dependency in the path has changed. Thus, we need to recompute only the rule dependency between rules in ΔR and the other rules in the rule base. If this remains the same, no path re-extraction is required. If it is different, then we need to focus only on the subset of paths that are affected by the change in the dependency. In Figure 6, we consider four cases: addition (deletion) of atoms in a rule consequent (antecedent). It outlines the heuristics for quick path extraction to facilitate testing whenever changes involve non-goal atoms (the underlying assumption being that non-goal atoms would be added/deleted more frequently than atoms used to infer goals).

All test conclusions based upon the above changes would be correct since they preserve the rule dependency in a path (warning the developer as appropriate), and ensure that all rules in a path are ready to fire once the goals required by a path are inferred. A general method for path extraction involving unconstrained rule modification that also takes into account modifications to the goal specification is an open problem.

Summary and Conclusion

Our aim to integrate the evaluation processes in a system life-cycle is based on providing a link that connects the conceptual, design, and implementation levels of a system. Knowledge is acquired using goal specification to capture the desired problem solving states. Based on the specified goals, the structure of the system at the implementation level is defined by a set of rule sequences inferring goals. Goals also play an important role in providing design alternatives based on the choice of goal composition versus hypotheses in a rule base (Chander, Radhakrishnan, & Shinghal 1997).

The design and development of rule-based systems often cause anomalies in the rule base: not only the development tends to be error prone, but there is confusion in applying evaluation processes for these systems (Hamilton, Kelley, & Culbert 1991). In this regard, our work emphasizes the following: (1) the system structure should play a ma-

1. **Addition of an atom in a rule antecedent** Every path that contains the rule requires re-extraction as the reachability[a] of this rule has changed. However, if the atom added in the antecedent is inferred in that path, do not re-extract. If the atom is an initial evidence or if it is a goal atom contained in the goals required for this path, then no re-computation is required. For example, addition of atom e in rule 4 does not cause any path re-extraction.
2. **Deletion of an atom in a rule antecedent** For saving computation, paths that contain this rule r need not be re computed as deletion does not affect the reachability of this rule from the other rules in the path (but the deleted atom and dependency due to this atom in the path should be noted). Of course, if the deletion causes a rule antecedent to become empty, or a path to get disconnected, report this as potential redundancy. For example, deletion of b in the antecedent of rule 2 makes path $\Phi 1$ disconnected. This should be reported, but the path need not be re-extracted (since its goal would be inferred any way), but rule 1 is now redundant as it makes a useless inference.
3. **Addition of an atom A in a rule consequent** If the rules reachable from this rule r are unchanged, then no computation is required for path extraction. Otherwise, for every new rule r' in path Φ' now reachable from r, one new path inferring the same goal g' as Φ' is added. In the new path r replaces a rule (or a set of rules) from Φ' that infers atom A. Note, adding new paths entail little computation: copy paths such as Φ' and simply perform a rule replacement. For example, adding d to the consequent of rule 1 adds a path inferring goal f containing rule 1 replacing rule 3 (in path $\Phi 2$).
4. **Deleting an atom in the consequent of a rule** If the rule dependency of a path is unaffected, do not re-compute. Otherwise, re-extract every path that contains this rule (because it can cause path disconnection). For example, deleting e from rule 3 causes a re-extraction of only path $\Phi 3$ and a subsequent report that rule 5 is redundant due to this change.

[a]Rule r_1 is reachable from rule r_2, if an atom in the consequent of r_2 unifies with an atom in the antecedent of r_1.

Figure 6: Incremental Path Extraction for changes involving addition or deletion of non-goal atoms. Examples correspond to the rule base shown in Figure 3.

jor role in system evaluation; (2) functional testing has its limitations; and (3) integrating evaluation should exploit incremental evaluation to cut cost and effort for testing without compromising on the quality of testing. We are currently working on the cost-benefit analysis for incremental path extraction and a general algorithm for incremental evaluation of rule-based systems. We hope the improvements for incremental evaluation on the existing tool suite to provide reliable test support for system developers.

References

Chander, P. G.; Radhakrishnan, T.; and Shinghal, R. 1997. Design Schemes for Rule-based Systems. *International Journal of Expert Systems: Research and Applications.* In press.

Chander, P. G. 1996. *On the Design and Evaluation of Rule-based Systems.* Ph.D. Dissertation, Department of Computer Science, Concordia University, Montreal.

Chang, C. L.; Combs, J. B.; and Stachowitz, R. A. 1990. A Report on the Expert Systems Validation Associate (EVA). *Expert Systems with Applications* 1(3):217–230.

Giovanni, G. 1989. Assuring Adequacy of Expert Systems in Critical Application Domains: A Constructive Approach. In Hollnagel, E., ed., *The Reliability of Expert Systems.* New York: Halsted Press. 134–167.

Grossner, C.; Gokulchander, P.; Preece, A.; and Radhakrishnan, T. 1996. Revealing the Structure of Rule-Based Systems. *International Journal of Expert Systems: Research and Applications* 9(2):255–278.

Guida, G., and Mauri, G. 1993. Evaluating Performance and Quality of Knowledge-Based Systems: Foundation and Methodology. *IEEE transactions in Knowledge and Data engineering* 5(2):204–224.

Hamilton, D.; Kelley, K.; and Culbert, C. 1991. State-of-the-Practice in Knowledge-based System Verification and Validation. *Expert Systems with Applications* 3(3):403–410.

Kiper, J. D. 1992. Structural Testing of Rule-Based Expert Systems. *ACM Transactions on Software Engineering and Methodology* 1(2):168–187.

Krause, P.; Fox, J.; Neil, M. O.; and Glowinski, A. 1993. Can We Formally Specify a Medical Decision

Support System? *IEEE Expert* 8(3):56–61.

Lee, S., and O'Keefe, R. M. 1994. Developing a Strategy for Expert System Verification and Validation. *IEEE Transactions on Systems, Man, and Cybernetics* 24(4):643–655.

Meseguer, P. 1992. Incremental Verification of Rule-based Expert Systems. In Neumann, B., ed., *10th European Conference on Artificial Intelligence*, 829–834.

O'Keefe, R. M., and Lee, S. 1990. An Integrative Model of Expert System Verification and Validation. *Expert Systems With Applications* 1(3):231–236.

O'Keefe, R. M., and O'Leary, D. E. 1993. Expert System Verification and Validation: A Survey and Tutorial. *Artificial Intelligence Review* 7(1):3–42.

Plant, R. T. 1992. Expert System Development and Testing: A Knowledge Engineer's Perspective. *Journal of Systems Software* 19(2):141–146.

Preece, A.; Grossner, C.; Gokulchander, P.; and Radhakrishnan, T. 1993. Structural Validation of Expert Systems: Experience Using a Formal Model. In *Notes of the Workshop on Validation and Verification of Knowledge-Based Systems (Eleventh National Conference on Artificial Intelligence)*, 19–26.

Shinghal, R. 1992. *Formal Concepts in Artificial Intelligence.* London, U.K., co-published in U.S. with Van Nostrand, New York: Chapman & Hall.