Int. J. Human–Computer Studies (1996) **44,** 145–169

Validating dynamic properties of rule-based systems

ALUN D. PREECE

Department of Computing Science. University of Aberdeen. King's College, Aberdeen, AB9 2UE, Scotland

CLIFF GROSSNER AND T. RADHAKRISHNAN

Concordia University, Computer Science Department, 1455 de Maisonneuve Boulevard Ouest, Montréal, Québec, H3G 1M8, Canada

Rule-based systems can be viewed as possessing two sets of properties: static and dynamic. Static properties are those that can be evaluated without executing the system, and dynamic properties can be evaluated only by examining how the system operates at run time. The dynamic properties of a rule-based system have been largely neglected in validation and verification work done thus far. Structural verification and static testing techniques do not yield information on how a rule-based system achieves its goals at run-time, given a set of input data. This paper presents a model for the relationship between the goal states achieved by a rule-based system, the set of inter-related rules that must fire to achieve each goal state, and the data items required for the rules in the rule sequence to fire. Then, we describe a method for applying this model to study the dynamic properties of a rule-based system. It is demonstrated that this model permits the validation of dynamic properties of a rule-based system, enabling system developers to decide: (1) if the manner in which the system pursues goals is valid according to the specifications (and expectations) of the designers; (2) what relationship exists between the quality of system output for a given test case and the goals achieved during problem-solving on that test case; and (3) how the overall problem-solving activity of the system relates to the availability of input data.

1. Validating rule-based systems

Rule-based systems have demonstrated their success as a software technology for solving ill-structured problems (Hayes-Roth, 1985). Characteristically, such problems are solved by a heuristic search through a problem space, represented by transitions between goal states (Simon, 1973; Rich, 1983). The rules in a rule-based system provide the knowledge for traversing the state space, and the knowledge which the system uses to decide which goals to pursue. Often, this problem-solving strategy must proceed with limited, incomplete information (as quantified by the number of available items of data). This is especially true of real time rule-based systems, which often cannot wait until all information is available before acting.

Although rule-based systems have been implemented successfully in many domains—including aerospace, manufacturing, business, and medicine—building these systems is a challenging task. In particular, assuring the reliability of these systems poses a difficult problem in exploiting this technology. The root of this

1071-5819/96/020145 + 25$12.00/0

problem lies in the ill-defined nature of the applications, and the novel nature of the technology. Since the applications are ill-structured, it is hard to define precise requirements for software solutions, leading to difficulties in determining whether the system meets its requirements (Batarekh, Preece, Bennett & Gogono, 1991). Moreover, since the architecture of rule-based systems is unlike conventional imperative software, many of the conventional software evaluation techniques cannot be adapted easily to these systems (Miller, Groundwater & Mirsky, 1993).

There has been considerable interest in the problem of assuring the reliability of rule-based systems for over a decade, manifesting itself in the development of a variety of techniques for the validation and verification (V&V) of rule-based systems (Gupta, 1990; Ayel & Laurent, 1991; Preece & Suen, 1993). Roughly speaking, validation is concerned with ensuring that a system meets its users' requirements, while verification is concerned with the integrity of the system itself; thus, validation subsumes verification. In considering the V&V problem, it is useful to distinguish two sets of properties of a rule-based system: static and dynamic. The *static* properties are those characteristics of a rule-based system that can be evaluated without its execution; these include the following.

(S.1) The goals (or goal states) of the rule-based system, as defined by the system designer.
(S.2) The inter-dependent sets of rules that must fire to achieve each goal.
(S.3) The metric for evaluating the quality of the result produced by the system.

The *dynamic* properties are those characteristics of a rule-based system that can only be evaluated by examining how the system operates at run time; these include the following.

(D.1) The goals that are achieved for a given set of test cases.
(D.2) The inter-dependent sequences of rules that fired for a given set of test cases.
(D.3) The quality of the result produced for a given set of test cases.

The vast majority of previous work in the area of V&V of rule-based systems concerns the verification and validation of static properties of the system. A large body of work addresses the structural verification of rule bases; many tools have been developed which are capable of demonstrating that a rule base is free from anomalies such as conflicting rules, redundant rules, and certain kinds of incompleteness (Preece, Shinghal & Batarekh, 1992; Nazareth, 1993; Ginsberg & Williamson, 1993; Meseguer & Verdaguer, 1993; Zlatareva & Preece, 1993). These anomalies are all static properties of the rule base of the system [special cases of property (S.2) above]. Another major body of V&V work concerns—in a broad sense—the testing of rule-based systems. These works include techniques aimed at ensuring that the testing received by a rule-based system is adequately thorough, based on coverage of goal states [property (S.1)] and rules fired [property (S.2)] (Rushby & Crow, 1990; Kiper, 1992; Kirani, Zualkernan & Tsai, 1992; Preece, Grossner, Gokulchander & Radhakrishnan, 1994), and metrics for quantifying the

quality of the output produced by the system [property (S.3)] (O'Keefe, Balci & Smith, 1987; Adelman, 1991; O'Keefe & O'Leary, 1993).

The dynamic properties of a rule-based system have been largely neglected in the V&V work done thus far. Structural verification and current testing techniques do not yield information on how a rule-based system achieves its goals at run-time, given a set of input data; therefore, there is no way to decide if the manner in which the system chooses to pursue goals is valid using these methods. Specifically, the following questions are not adequately addressed by previous work in V&V.

- Is the manner in which the system pursues goals valid according to the specifications (and expectations) of the system designers? This questions both the validity of the goals pursued during problem-solving for a given test case (D.1), and the validity of the sets of inter-related rules fired to achieve those goals (D.2).
- What is the relationship between the quality of system output for a given test case (D.3), and the goals achieved during problem-solving (D.1)? This questions both the validity of goals pursued (D.1),† and the validity of the metrics used to quantify output quality (D.3)
- How does overall problem-solving activity [(D.1), (D.2) and (D.3)] of a rule-based system relate to the availability of input data, and when does its performance degrade to an unacceptable level as fewer data items are available? This question is of particular concern to developers of rule-based systems which must reason with limited information, such as real-time systems (Laffey, Cox, Schmidt, Kao & Read, 1988) and individual agents in distributed systems (Grossner, Lyons & Radhakrishnan, 1992; Grossner, Preece, Gokulchander, Radhakrishnan & Suen, 1993).

The research described herein is aimed at giving rule-based system developers the means to study—and thus validate—dynamic properties. This paper describes our models, techniques and tools used to study dynamic properties, and demonstrates their use on a test-bed application. Section 2 describes an approach to modelling a rule-based system as an inter-dependent sets of rules which achieve goals; using this model, we show how to identify the data requirements for each goal (in terms of the data items needed to achieve each goal). Section 3 presents the application of the structural model to study the dynamic properties of a rule-based system. In Section 4, we demonstrate using a realistic ''test-bed'' rule-based system how dynamic properties can be tested. We show that the information provided to the rule-base designer by studying dynamic properties enhances the rule-base designer's ability to validate the design of a rule-based system over what can be accomplished with static techniques. The dynamic properties are validated by comparison with the expectations of the rule base designer, based upon the knowledge acquired from the domain expert(s). In Section 5, we discuss how the use of our structural model can be incorporated into the entire development process for a rule-based system, and that the testing of dynamic properties is also a part of this development process. Since,

† This is different from the first question because not all of the goals pursued have a direct effect on the final output offered by the system.

like most V&V techniques, our method involves a non-trivial amount of human effort to set-up and to run, we aim to demonstrate that the results are worth that effort.

The Blackbox Expert is the testbed we use in this paper. This rule-based system has been designed to allow us to simulate the environment that a rule-based system would experience when forced to operate with limited information, in terms of the data items that it may access when problem-solving. The rule base of the Blackbox Expert is sufficiently complex to demonstrate the efficacy of our approach (containing hundreds of rules), and serves as a source of realistic examples for this paper (Grossner, Lyons & Radhakrishnan 1991).

The Blackbox Expert is designed ro solve a puzzle called Blackbox, which is a diagnosis type problem (Grossner *et al.,* 1991). The Blackbox puzzle consists of an opaque square grid (box) with a number of balls hidden in the grid squares. The puzzle solver can fire beams into the box, and observe the beam's exit point, if it exists. The beams fired into the grid will interact in different ways with the balls hidden in the grid, and the puzzle solver's goal is to determine the contents of the box based on the entry and exit points of the beams. The puzzle solver must determine if each location of the grid square is empty or contains a ball, and in addition if the conclusion drawn for the location is certain. Blackbox is an ill-structured problem; when the puzzle solver selects a beam to be fired, the outcome of firing the beam is not known. Blackbox is also an example of a problem where partial solutions are possible.

2. Modelling rule-based systems

Validating the dynamic properties of a rule-based system requires an understanding of the sequence of rules that must fire to achieve each goal. We call these interdependent sets of rules *paths* (Grossner *et al.,* 1993). In Section 2.1, we describe our model for determining the paths of rules that will fire when a rule-based system is problem-solving, where each path is related to the goal state achieved. In Section 2.2, we show how we can determine the data items required by each path in order to fire all the rules it contains; thus, paths permit us to relate data items required by the rule-based system to the goals it achieves.

2.1. RULE BASE STRUCTURE

Viewed in terms of a state space model (Rich, 1983), a rule-based system attempts to progress from state to state, selecting at each point the next desirable state based on the current information available. Each state may be a final state, or an intermediate state representing a ''meaningful'' advancement in reaching a final state. It is these states (intermediate and final) that we refer to as *goal states.*

We define a rule-based system $\mathscr{E}$ by means of the triple $\langle E, RB, WM \rangle$ where E is an inference engine, RB is the set of rules for solving the problem, and WM is the working memory where current data, represented by *facts,* are stored. The state of $\mathscr{E}$ at any given time is provided by the set of facts present in WM at that time. A fact

that may be present in *WM* is denoted by $f = \langle \lambda, l \rangle$ where λ is a predicate, l is a list of data elements, and λ identifies the relationship between the elements of l.

In modelling a rule-based system for dynamic validation, we need to be able to determine if a goal state has been reached: this indicates that a meaningful advancement in solving the problem has occurred. For this purpose, we assume that the rule base designer (implicitly in conjunction with the domain expert) is able to specify abstract goal states for the problem to be solved. This is done initially by examining the predicates used in the rule base, and indicating which of them constitute part of some goal state (intermediate or final); these predicates are referred to as *end predicates.* We use Z to denote the set of end predicates. Then, each goal state is defined by a conjunction of selected end predicates, called a *logical completion.* To relate goal states to rules, we use the notation $S \rightsquigarrow U$ to indicate that a set of rules U assert facts using all the predicates of some `logical completion` for a problem S. Thus, when the rules in U fire, a goal state of problem S will have been reached.

In order to relate the actual rules of the rule base to the goal states in the model, we require a suitable abstraction for rules. These *abstract rules* are ultimately used to construct the paths that we use to model rule-base structure. An abstract rule r consists of an LHS and a RHS where:

- the LHS is a set of fact *templates,* denoted by $\mathscr{I}^r$, such that at least one fact *matching* each template must be present in *WM* for the rule to fire;
- the RHS indicates the set of facts that are asserted by r, denoted by $\mathscr{A}^r$.

Essentially, a template is a specification for a type of fact that must be present in the *WM* in order for a rule to become enabled to fire; as such, each template $t = \langle \Lambda, L \rangle$ consists of a specification for a predicate Λ and a list of variables L. A fact is said to match a template if there exists a most general unifier δ such that $\langle \Lambda, L \rangle \cdot \delta = \langle \lambda, l \rangle$. For convenience, we define a mapping function $\mathscr{V}: A \rightarrow B$, where $A = \{(\Lambda_1, L_1), \ldots, \langle \Lambda_n, L_n \rangle\}$ is a set of templates or facts, $B = \{\lambda_1 \cdot \cdot \cdot \lambda_n\}$ is a set of predicates, and λ_j satisfies the specification for Λ_i. For simplicity, $\mathscr{V}(\{\Lambda_i, L_i)\}) = \{\lambda_j\}$ will be denoted by $\mathscr{V}(\langle \Lambda_i, L_i \rangle) = \lambda_j$.

The templates and facts as well as the mapping function $\mathscr{V}(\langle \Lambda_i, L_i \rangle)$ gives us a notation we can use to develop a notion of inter-rule dependency, which we need to define paths. One rule is dependent upon another if the action taken by one rule "permits" the other rule to become fire-able. The conventional form of inter-rule dependency is where one rule asserts a fact on its RHS which is required by the LHS of another rule. Formally: $r_i \prec r_j \equiv (\exists \langle \lambda, l \rangle \in \mathscr{A}^{r_i},\ \lambda \notin Z) \Rightarrow (\exists \langle \Lambda, L \rangle \in \mathscr{I}^{r_j}, \mathscr{V}(\langle \Lambda, L \rangle) = \lambda)$. Note that we restrict inter-rule dependency: the condition $\lambda \notin Z$ placed on the *depends upon* relation restricts the dependency relationship to rules asserting facts which do not identify goal states. In this way, we ensure that rule sequences that are constructed using the `depends upon` relation (i.e. paths) will terminate when a goal state is reached.

Having defined the `depends upon` relationship between two rules, methods for grouping rules to form *continuous* sequences must be considered, that is, the rule sequences must not stop before a goal state is reached. We capture a continuous sequence of rules by defining a notion of *closure* on a set of rules under the

`depends upon` relationship. When considering a set of rules Φ, Φ is *closed* for *depends upon* if every rule in Φ asserts a fact either using an end predicate, or a predicate matching a template on the LHS of another rule in Φ.

For the purposes of validation, we want to ensure that each path leads to the reaching of a single, identifiable goal state, and that it represents no more than one way to reach that goal. In other words, we want out paths to be *unambiguous.* In order to remove ambiguities from rule sequences, we introduce a restriction called *singular consumption.* The restriction of `singular consumption` on a set of rules Φ ensures that every predicate that is asserted by a rule in Φ participates in exactly one `depends upon` relationship: this prevents any rule contributing either to the reaching of more than one goal in a single path, or to the reaching of the same goal in more than one way. (The rule can still act in this fashion in the structural model as a whole, but it must do so in *different* paths.)

Rule sequences constructed using the `depends upon` relation, that are `closed`, *and that are* `singularly consumed` are still not quite sufficient to define a path for our purposes: we must also account for all facts that are needed to enable each rule in the sequence to fire. A set of rules W *enables* a rule r to fire if W satisfies the following conditions:

- r must `depend upon` every rule in W;
- every rule in W must assert at least one fact that uses a predicate specified by the LHS of r, where a fact using that predicate is not asserted by any other rule in W;
- for each predicate specified by a template on the LHS of r, if that predicate is used in a fact asserted by at least one rule, then some rule that asserts a fact using the predicate must be a member of W. $W \hookrightarrow r$ denotes that W is an enabling-set for r.

The rule sequences we construct using all of the above conditions permit us to model a rule base as a set of paths. Each path relates the structure of the problem (in terms of goal states and logical completions) to the artifacts used to solve the problem (the set of inter-dependent rules). Our formal definition for a path is as follows.

Definition 1 (Path). *P_k, a path k in rule base RB solving problem S is a partially-ordered set of rules $\langle \Phi, \pi \rangle$.*

Φ is a set of rules $\{r_1, r_2 \cdots r_n\}$ with $r_i \in RB$ such that,
$(\exists U \subseteq \Phi,\ S \rightsquigarrow U)$,
$(\forall r_i \in \Phi)(\exists W \hookrightarrow r_i,\ W \subset \Phi)$,
Φ is `closed` *under* `depends upon`*, and*
Φ is `singularly consumed`*.*

Path Hunter is a rule base analysis tool we have constructed to determine the paths contained in a rule base (Grossner, Gokulchander, Preece & Radhakrishnan, 1993). Given a rule base and the set of declarations for the logical completions associated with each of its sub-problems, the tool automatically creates the set of

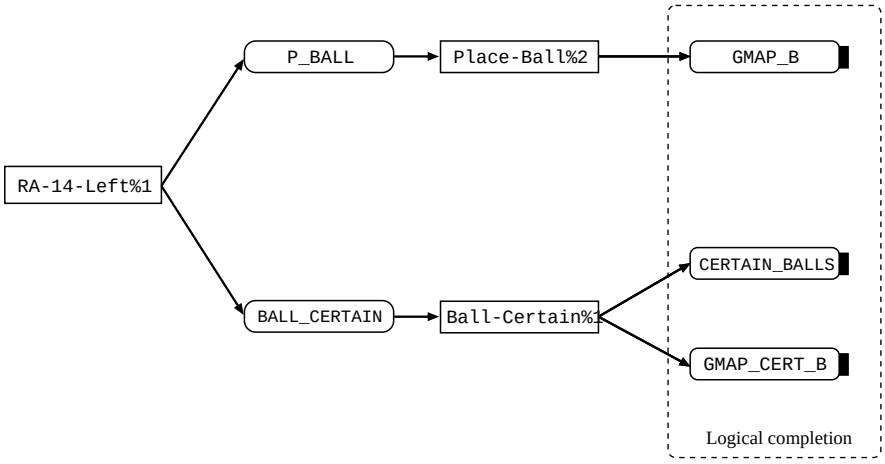


FIGURE 1. An example path.

abstract rules, and generates the set of all paths in the rule base. Path Hunter found 512 abstract rules in the Blackbox Expert's rule base, which formed 516 paths. A single path found by Path Hunter is shown in Figure 1. The square nodes represent abstract rules, the ''round'' nodes represent predicates, and the directed arcs represent data dependencies between the rules. The *logical completion* that is asserted by this path is GMAP_B $\wedge$ CERTAIN_BALLS $\wedge$ GMAP_CERT_B. The semantics for this path are as follows: upon examining the evidence (beam entry and exit points) currently available, select the actions of placing a ball on the grid and marking the ball as certain; and the outcome is, a ball is placed and this ball is marked as certain.

2.2. DATA REQUIREMENTS

The data requirements of a rule-based system can be determined using our formal model for capturing rule-base structure. We recall that a path is a set of rules that advance the state of the problem being solved from one goal state to another. The facts required by the rules in a path to fire are the precedence constraints for advancing the state of the problem being solved. There are two issues in using the path model to determine the data requirements of a rule-based system: determining the precedence constraints required for a single rule in a path to fire, and determining when all the rules in a path will be able to fire.

The initial set of rules that must fire in a path are called *start rules.* Start rules will then assert facts enabling the other rules in the path. The start rules of a path P_k, denoted by SR_k, is the set of rules $r_i \in \Phi$ where the templates in the LHS of r_i do not *depend upon* any other rule $r_j \in \Phi$; $SR_k = \{r_i \mid r_i \in \Phi,\ (\forall r_j \in \Phi,\ r_i \not\prec r_i)\}$. For start rules, $W = \varnothing$.

The *start set* of a path P_k is the set of facts required by the start rules of that path to become enabled to fire. A start set of a path is identified by the set of *start templates,* denoted by ST_k, which is the union of the set of templates present on the LHS of each of the start rules in the path; $ST_k = \cup_{r_i \in SR_k} \mathcal{I}^{r_i}$. The set of predicates

used in the start templates for a path is given by $SP_k = \{\lambda_j \mid \forall\langle\Lambda_i, L_i\rangle \in ST_k, \mathcal{V}((\Lambda_i, L_i)) = \lambda_j\}$.

The *completion set* of a path is the set of facts required for all the rules in that path, except start rules, to become enabled to fire. A completion set of a path is identified by the set of *completion templates,* and is denoted by CT_k. The completion templates is the set of all templates present on the LHS of all the rules of the path, except start rules, which specify a fact that is not asserted by any rule in the path. Formally, $CT_k = \bigcup_{r_i \in (\Phi - SR_k)}(\mathcal{I}^{r_i} - PT_{i,k})$. $PT_{i,k}$ is the set of templates from $\mathcal{I}^{r_i}$ that are matched by the facts asserted when the rules that enable r_i fire. $PT_{i,k} = \{\langle\Lambda_i, L_i\rangle \mid (\forall r_j \in W,\ W \hookrightarrow r_i)(\exists\delta, \langle\Lambda_i, L_i\rangle \cdot \delta = \langle\lambda_i, l_i\rangle, \langle\lambda_i, l_i\rangle \in \mathcal{A}^{r_i})\}$. The set of predicates used in the completion templates for a path is given by $CP_k = \{\lambda_j \mid \forall\langle\Lambda_i, L_i\rangle \in CT_k, \mathcal{V}(\langle\Lambda_i, L_i\rangle) = \lambda_j\}$.

Now, let us consider the precedence constraints for a rule in a path. Given a path P_k, $r_i \in \Phi$, $r_i \notin SR_k$, and $W \hookrightarrow r_i$, then the set of completion templates for r_i is given by $CT_{i,k} = (\mathcal{I}^{r_i} - PT_{i,k})$.

Lemma 1. *If facts matching the completion templates $CT_{i,k}$ of rule r_i in a path P_k are present in WM, $r_i \notin SR_k$, and the $(r_j \in W,\ W \hookrightarrow r_i)$ fire, then r_i can fire.*

Proof. After the $r_j \in W$ fire, *WM* contains the union of all facts asserted by the $r_j \in W$, in addition to the facts matching $CT_{i,k}$. These facts match the templates $CT_{i,k} \cup PT_{i,k}$ by definition. This simplifies to $\mathcal{I}^{r_i}$ (by putting $CT_{i,k} = (\mathcal{I}^{r_i} - PT_{i,k})$); thus, the facts in *WM* match all the LHS templates of r_i, and hence r_i can fire.

Now that we have understood the conditions required for an individual rule in a path to fire, we can determine the conditions for all the rules in a path to become to fire-able.

Theorem 1. *Given a path P_k, if facts matching the completion templates CT_k and start templates ST_k are present in WM, then all the rules in the path can fire.*

Proof. The proof of the theorem follows directly from Lemma 1 by induction.

According to Theorem 1, ST_k and CT_k identify the data items will be required by the rule-based system to achieve its goals. In Theorem 1, we have shown that the precedence constraints for each path, ST_k and CT_k, indicate the facts required for all the rules in that path to fire, achieving a goal; the goal achieved is identified by the logical completion asserted by the rules in the path.

Our Path Hunter tool provides the data requirements for each path (ST_k and CT_k). For example, the data requirements for our earlier example path are shown in Figure 2. The set of start rules for the example path shown in Figure 1 is {`RA-14-Left%1`}. The data items required by this path to achieve the goal represented by the logical completion it asserts are given by its start predicates {`GMAP, SHOT-RECORD, GRIDSIZE`}, and its completion predicates {`GMAP_CERT, CERTAIN_BALLS`}. The start predicates for this path indicate that this path will access facts the indicate the contents of the grid squares, the beams that have been fired. The completion predicates indicate that this path requires access to facts that indicate the certainty of the hypothesis for the contents of the grid squares that have been identified.

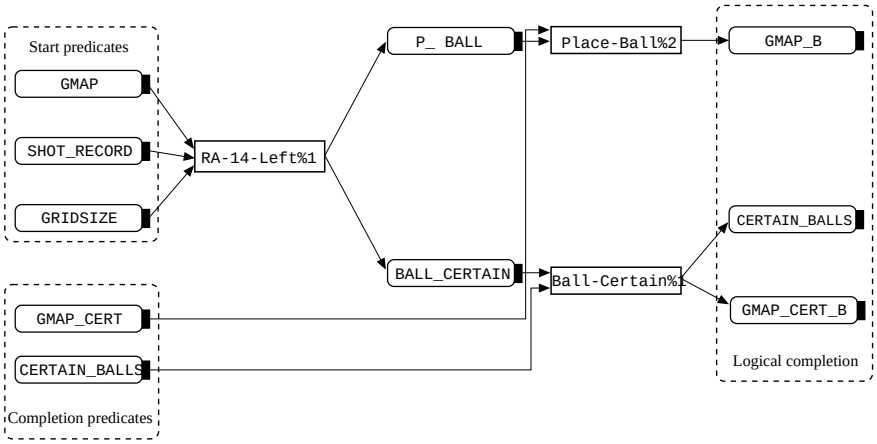


FIGURE 2. An example path, showing its data requirements.

3. Measuring and validating dynamic properties

Having developed a structural model in terms of paths, let us now consider how we can study the dynamic properties of a rule-based system in order to answer the questions we raised earlier in this paper.

- Is the manner in which the system pursues goals valid according to the specifications (and expectations) of the system designers?
- What is the relationship between the quality of system output for a given test case, and the goals achieved during problem-solving?
- How does overall problem-solving activity of a rule-based system relate to the availability of input data, and when does performance degrade to an unacceptable level as fewer data items are available?

In studying dynamic properties we assume that there is an existing rule-based system $\mathscr{E}$, and Path Hunter has been used to analyse the rule base of $\mathscr{E}$, giving the set of paths $\{P_k \mid P_k \text{ is a path in } \mathscr{E}\}$ as well as the start and completion predicates for each path; SP_k and CP_k. Our exploration of dynamic properties will require that we investigate the relationship between the data items available to a rule-based system, the goals it achieves, and the result it produces.

By its very nature, the study of dynamic properties implies that an operational system is to be exercised; thus, the study of dynamic properties is experimental in nature. Experimentation with any system requires careful consideration of different requirements. Of primary importance is the type of experiments that will be carried out; that is, the hypotheses that will be tested in the experiments, the input parameters required for conducting the experiments, and the outputs that are to be measured during the experiments.

In this section, we first describe a design for experiments which can be performed to test various hypotheses concerning dynamic properties; then, we present a method for conducting these experiments. As with the static analysis of a rule-based system, our structural model plays an important role in permitting the knowledge

engineer to conduct experiments designed to test the dynamic properties of rule-based systems.

3.1. DESIGN OF THE EXPERIMENT

The design of an experiment typically starts with the hypotheses to be tested. Then, using the hypotheses one must consider the parameters of the system under study that are to be varied, those parameters that are to be inputs to the system, and those that are to be measured as outputs. Once the hypotheses, inputs and outputs that will be considered in the experiment have been determined, the exact procedure to be followed during the experiment can be identified. In the design of experiments for dynamic validation of rule-based systems, we will first consider the types of hypotheses that might be selected by the knowledge engineer, the inputs to the system, and the outputs that should be measured. Then, we will discuss the procedure to be followed.

The types of hypotheses that we have identified as pertinent for studying dynamic properties of rule-based systems are as follows (other hypotheses may be identified using the framework we provide in this paper, which may be of interest for specific applications).

(H.1) Hypotheses concerning the goals that are achieved (expected behaviour) for a specific test case (or set of test cases), given the data items available. For example, the rule base designer may postulate that a specific goal would be achieved for a given test case and set of data items being available, and then indicate that a different goal is expected to be achieved as the set of data items available is modified.

(H.2) Hypotheses concerning the paths that are pursued as well as completed by the rule-based system for a specific test case (or set of test cases), given the data items available. For example, the rule base designer may postulate that for a given test case and set of data items being available, a goal would be achieved by a *specific path,* or that a set of paths would become activated because the data items in their start set would be present in the working memory of $\mathscr{E}$.

(H.3) Hypotheses concerning the goals achieved by the rule-based system and the quality of the result produced. For example, the rule base designer may postulate that for a given test case and set of data items being available, that the achievement of a specific goal would improve the quality of the result that was produced.

(H.4) Hypotheses concerning the relationship between the available data and the quality of the result produced. For example, the rule base designer may postulate that for a given set of data items being available the rule-based system will produce results of a specific quality.

Given the hypotheses to be tested, we can clearly see that the inputs that are required for the experiment are a set of test cases and a set of expected behaviours for the rule-based system. The variable in these hypotheses is the data items that are to be made available to the rule-based system as it solves each test case. Each time a test case is solved, we must be able to measure the goals that are actually achieved by the rule-based system, the paths pursued, the paths that were completed, and the quality of the result produced.

Let us refer to the set of test cases to be used in the experiment as $\mathscr{T}$. The domain expert may choose to select test cases randomly so that there is no bias (on the part of the rule-base designer) added to the experiment, or select specific test cases to check scenarios that have been identified as potential problem areas for the system, or by using the structural model itself to ensure that the set of cases will exercise the components of the system sufficiently completely (Preece, Grossner, Gokulchander & Radhakrishnan, 1994). When considering dynamic behaviour for a specific test case, we must consder both the goals which are expected to be achieved for each test case and paths pursued. We will denote the goals which are expected to be achieved for each test case by $(M_e, 1)$ [this corresponds to dynamic property (D.1)], and denote the paths which are expected to be pursued by $(M_e, 2)$ [this corresponds to dynamic property (D.2)]. In any case, these expectations and test cases need to by provided by the system builder(s) and domain expert(s); this is discussed further in Section 5. The rule-base designer must also determine the different sets of data items that are to be made available to the rule-base system as the different test cases in $\mathscr{T}$ are solved; the sets of data items are denoted by (V.1). We will denote the actual goals achieved by (M.1), the paths pursued by (M.2), the path completed by (M.3), and the quality of the result produced by (M.4).

The experiments that we propose for validating dynamic properties will require that the rule-based system $\mathscr{E}$ solves every test case in the set $\mathscr{T}$ for each set of data items contained in (V.1). The measurements taken will be the actual goals achieved, (M.1); the paths pursued, (M.2); the paths completed, (M.3); and the quality of the result produced, (M.4). Performing experiments of this type would enable us to test each of the four different types of hypotheses: (H.1) can be tested by comparing (M_e.1) with (M.1); (H.2) can be tested by comparing (M_e.2) with (M.2), or (M_e.2) with (M.3); (H.3) can be tested by comparing (M.1) with (M.4); and (H.4) can be tested by studying the changes in (M.4) that occur as the test cases in $\mathscr{T}$ are solved with the different sets of data items specified in (V.1).

3.2. EXPERIMENTAL METHOD

Experiments following the design we propose to validate dynamic properties will be conducted by having the rule-based system $\mathscr{E}$ solve every test case in the set $\mathscr{T}$ m times, where m is the number of sets of data items contained in (V.1). For n test cases this requires $n \times m$ executions of $\mathscr{E}$, denoted by $X_1 \cdots X_{(n \times m)}$. Each execution X_i is subsequently analysed to yield the measurements:(M_i.1), the goals achieved in execution X_i; (M_i.2) the paths pursued in execution X_i; (M_i.3), the paths completed in execution X_i; (M_i.4), the quality of output from execution X_i. In order to conduct this type of experiment we need mechanisms for measuring (M_i.1), (M_i.2), (M_i.3), and (M_i.4), as well as a method for quantifying the level of ''information'' supplied to the rule-based system by each data items contained in (V.1). We will now consider methods for obtaining each of these measurements and a method for quantifying the level of ''information'' supplied to the rule-based system by the sets of data items in (V.1).

Quality of result produced: In order to evaluate the *quality* of the solution represented by the achievement of a set of final goal states, we need an appropriate metric or judgment standard. This is necessarily domain-dependent, and may be a quantitative or qualitative measure.

Information available: In order to allow us to compare measurements (M_i.1), (M_i.2), (M_i.3), and (M_i.4) that will be obtained for each set of data items in (V.1), we need a means of quantifying the ''amount'' of information that is provided to the rule-based system by a particular set of data items. The ability to quantify the amount of information provided will also help us to quantify data availability that are to be used in the experiment. We make use of a metric for the availability of data items to rule-based system, developed by Grossner, called the information deficit metric (Grossner, 1994). The information deficit metric produces a value between zero and one, estimating the availability of the data items that are required by the rule-based system when problem-solving. An information deficit of zero indicates that all the data items are available to the system.

Goals achieved, paths pursued and paths completed: The goals achieved (M_i.1), the paths pursued (M_i.2), and the paths completed (M_i.3) are determined by applying the structural model we described in Section 2 to analyse trace files produced by the rule-based system as it solves each test case in $\mathscr{T}$. (M_i.2) is the set of paths in which at least one rule can be determined to have fired while $\mathscr{E}$ solved the test case at hand. (M_i.3) is the set of paths in which all rules can be determined to have fired. (M_i.1) is given by the logical completion asserted by all paths in (M_i.3) (remember that each logical completion corresponds to a goal state). Normally, a trace of the inter-dependent sequences of rule firings which occurred when the system was run can be obtained by an automatic analysis of the information most commercial inference engines are capable of producing for debugging, but even if the inference engine is incapable of providing such information automatically, it is possible to instrument the rule base to create a trace as a side-effect of normal execution.

Since each path P_k is a partially-ordered set of abstract rules, and the run-time trace is a partially-ordered set of ''real'' rule firing events, it is necessary in analysing trace files to identify correspondences between the abstract rules and their ''real'' or *concrete* counterparts. If an unambiguous one-to-one correspondence can be made, then it is a straightforward matter to determine the abstract rule that correspond to a concrete rule and to determine the path to which the concrete rule belongs. In reality, there may exist ambiguities in determining to which path a rule firing sequence belongs, because it is not always possible to decide unequivocally which abstract rule corresponds to a given conrete rule firing observed in the trace. It is clear why this may be so when the abstract rules are considered as specifications for the form of the concrete rules: this admits the possibility that, due to a fault in the rule base coding, a concrete rule may not conform to its abstract specification. In some cases, when a rule fires not all of the facts it is designed to assert because the facts are already present in working memory; thus, the concrete rule does not match its abstract specification. When a concrete rule does not match its specification, it may appear to correspond to zero or more abstract rules (this phenomenon is elaborated further in (Preece *et al.,* 1994)). We refer to the case where a concrete rule firing appears to correspond to more than one abstract rule as an *equivocal mapping.*

Equivocal mappings force us to use different strategies for determining the abstract rule that correspond to a concrete rule and the path to which the concrete rule belongs. With equivocality, we cannot be sure if some of the rules in a path have or have not actually been observed in the trace; therefore, we employ three

different measuring strategies: conservative, moderate, and liberal. To formalize our measurement method: abstract rules are denoted as before by r_i; concrete rule firings are denoted by $\mathbf{r}_j$; the fact that rule $\mathbf{r}_j$ causes rule $\mathbf{r}_k$ to fire at run-time is denoted by $\mathbf{r}_j \gg \mathbf{r}_k$; unequivocal (i.e. unambiguous) mappings are denoted by $\mathbf{r}_j \mapsto r_i$; and equivocal mappings are denoted by $\mathbf{r}_j \overset{?}{\mapsto} r_i$.

We use the notion of a *thread* to determine which rules in path have fired: there is a thread from rule r_1 to rule r_n in a path if the path contains the set of `depends upon` relations $\{r_1 \prec r_2, r_2 \prec r_3, \ldots, r_{n-1} \prec r_n\}$. The threads in the example path in Figure 3 are; $\{d_1, d_2\}$, $\{d_1, d_3, d_5\}$ and $\{d_1, d_4, d_6\}$. A thread is *observed* if and only if there is a set of mappings $\{\mathbf{r}_1 \mapsto r_1, \mathbf{r}_2 \mapsto r_3, \ldots, \mathbf{r}_{n-1} \mapsto r_{n-1}, \mathbf{r}_n \mapsto r_n\}$, and there is a set of firing causalities $\{\mathbf{r}_1 \gg \mathbf{r}_2, \mathbf{r}_2 \gg \mathbf{r}_3, \ldots, \mathbf{r}_{n-1} \gg \mathbf{r}_n\}$.

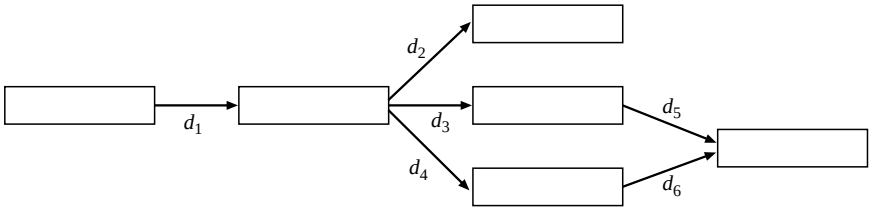
To illustrate this, assume in the example that there are unequivocal mappings to *all but the rightmost rule.* Therefore, observed threads in the example path are: $\{d_1, d_2\}$, $\{d_1, d_3\}$, $\{d_1, d_4\}$. The thread $\{d_1, d_3, d_5\}$ is not observed because there is no unequivocal mapping to the rightmost rule in dependency d_5. Similarly, thread $\{d_1, d_4, d_6\}$ is not observed because there is no unequivocal mapping to the rightmost rule in dependency d_6.

We determine the number of rules that have fired in a path by examining dependent pairs of rules in the run-time trace; whether a dependent pair of rules is accepted or not as being in a path depends upon their position in a thread, relative to the position of equivocal mappings. Each of our three measurement strategies considers equivocal mappings differently for the purposes of accepting pairs of dependent rules.

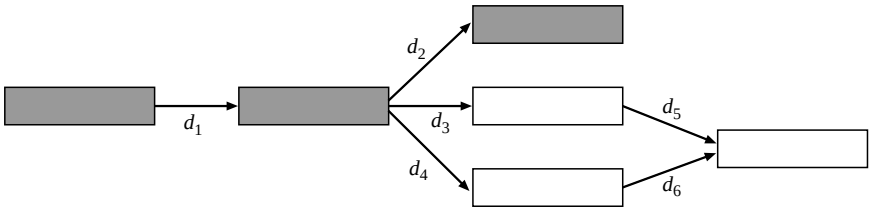
In the *conservative strategy,* we count all the dependent pairs in each thread only if all are present in the trace, and no equivocal mappings are involved. More formally, we count the dependent pair $r_i \prec r_j$ as having been observed if and only if there is an observed thread from r_s to r_e, where r_s is a start rule for the path, r_e is an end rule for the path, and the thread contains $(r_i \prec r_j)$. Using this strategy, only the two dependent pairs comprising the example thread (d_1, d_2) are counted, because all other threads from a start rule to an end rule involve an equivocal mapping [Figure 3(b)]. The conservative strategy thus minimizes the potential that a dependent pair in a path was accepted as having been present in a trace file, when in fact it was not.

Both the moderate and liberal strategies accept dependent pairs in each thread as far as the first dependent pair which contains an equivocal mapping. In the moderate strategy, dependent pairs containing an equivocal mapping *are not* counted, but in the liberal strategy, dependent pairs containing an equivocal mapping *are* counted. Thus, in the *moderate strategy,* we accept dependent pair $r_i \prec r_j$ if and only if there is an observed thread from r_s to r_j, where r_s is a start rule for the path, and the thread contains $(r_i \prec r_j)$. In the example (d_1, d_2, d_3, d_4) are counted [Figure 3(c)]. In contrast, in the *liberal strategy,* we accept $r_i \prec r_j$ if and if only if there is an observed thread from r_s to r_i (where r_s is a start rule for the path), $\mathbf{r}_i \mapsto r_i$, $\mathbf{r}_i \gg \mathbf{r}_j$, and $\mathbf{r}_j \overset{?}{\mapsto} r_j$. Using this strategy, all six of the dependencies in the example are counted [Figure 3(d)].

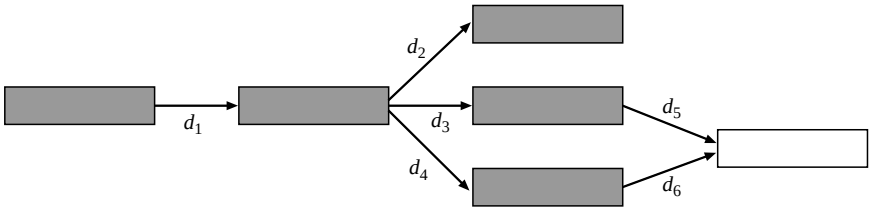
Determination of goals achieved (M_i.1), paths pursued (M_i.2), and paths completed (M_i.3) is practical only if it is performed automatically: we have constructed Path Tracer, an analysis tool that examines the trace files produced by a



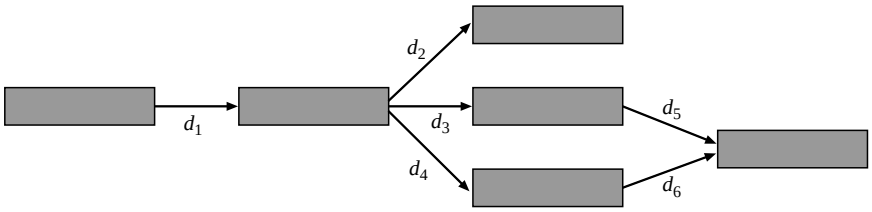
(a) The example path



(b) Coverage of the path using conservative strategy



(c) Coverage of the path using moderate strategy



(d) Coverage of the path using liberal strategy

FIGURE 3. Examples of coverage in an example path. (a) The example path; (b) coverage of the path using `conservative` strategy; (c) coverage of the path using `moderate` strategy; (d) coverage of the path using `liberal` strategy.

rule-based system whcn it is problem-solving, to determine the number of rules that have fired in each path (Preece *et al.,* 1994). Path Tracer was implemented to process trace files produced by the inference engine of the CLIPS production system (Giarratano & Riley, 1989), but the basic mechanism we use to measure (M_i.1),

(M_i.2), and (M_i.3) in this paper is general, and it is not difficult to modify Path Tracer to analyse trace file produced by inference engines other than that of CLIPS.

4. Validating dynamic properties: a case study

In this section, we demonstrate the use of the path model, Path Hunter, and Path Tracer in studying the dynamic properties of the Blackbox Expert. Our demonstration follows the design presented in Section 3; the Blackbox Expert solved a set of 10 test cases, and each test case was solved five times. Each time the same test case was solved, we varied the data items available to the Blackbox Expert. Random test cases were used because we did not want to bias our experiment, and our goal is to demonstrate the benefits of validating dynamic properties of rule-based systems, not to exhaustively validate the dynamic properties of the Blackbox Expert. We quantified the data items available when a test case was solved using the information deficit metric. When the Blackbox Expert solved each test case, we measured the actual goals achieved, (M.1); the paths pursued, (M.2); the paths completed, (M.3); and the quality of the result produced, (M.4). The quality of the result produced by the Blackbox Expert is measured using the domain dependent metric called SCORE, developed by human experts in solving Blackbox puzzles (Grossner *et al.,* 1991). The SCORE metric assigns a numerical value between 0 and 300 to a proposed solution for a Blackbox puzzle, based upon the number of grid squares that are correctly identified and the number of beams fired to solved the puzzle. The lower the value assigned by the SCORE metric, the better the proposed solution. The goals that were achieved by the Blackbox Expert each time a test case was solved were determined using Path Tracer.

In our demonstration we will consider the following aspects of the Blackbox Expert's dynamic properties.

- We will determine how the problem-solving ability of the Blackbox Expert—both in terms of goals achieved (D.1) and output quality (D.3)—relates to the availability of input data; this is determined by comparing the number of goals that are achieved (M.1) with the SCORE (M.4) of the results produced as the data items available are reduced. Examining this relationship provides a mechanism by which we can decide when performance can degrade to an unacceptable level, as fewer and fewer data items are available.
- We will determine the relationship between the quality of the output produced by the Blackbox Expert for a given test case [property (D.3)], and the goals it achieved solving that test case (D.1); thus, we will establish the impact of achieving a specific goal on the result produced. Determining this relationship is accomplished by considering each goal that is achieved (M.1) as each test case is solved, the path that was responsible for the goal being achieved (M.3), and the SCORE of the result produced (M.4).
- We will examine the manner in which the Blackbox Expert pursues goals and decide if this is valid according to its specifications (and the expectations of its designers). We base this decision both upon the system's choice of goals to pursue at run-time [property (D.1)], and upon the paths it activates to achieve these goals [property (D.2)]. We determine the system's choice of goals by

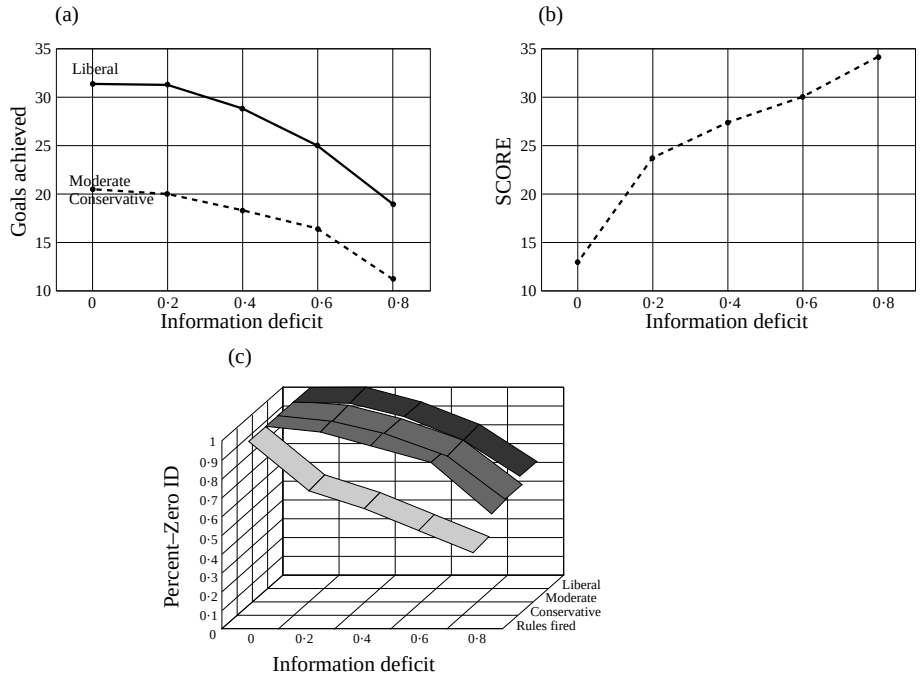


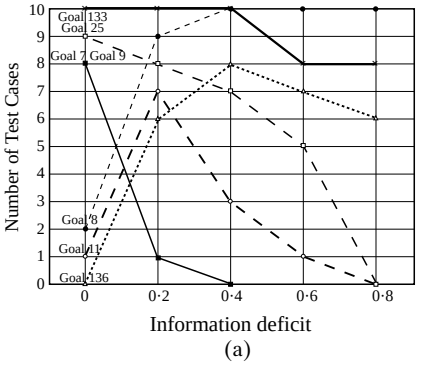
FIGURE 4. Goals achieved, SCORE, and rules fired.

examining goals achieved (M.1) and the paths activated in achieving goals (M.2) and (M.3); it is also possible that a decision as to the acceptability of a goal being pursued would require consideration of SCORE (M.4).

Problem-solving ability: Let us first consider how the problem-solving ability of the Blackbox Expert was affected by the availability of data. Figure 4(a) indicates the total number of goals that were achieved by the Blackbox Expert as it solved the test set, and Figure 4(b) shows the average SCORE for the results produced by the Blackbox Expert as the information deficit increases. As expected, the SCORE of the results produced by the Blackbox Expert increased as the information deficit it faced increased, and the number of goals achieved under each counting strategy decreased as the information deficit faced by the Blackbox Expert increased. This indicates that the ability of the Blackbox Expert to solve Blackbox puzzles is reduced as the number of data items available is reduced.

The shape of the curves shown in Figure 4(a) and Figure 4(b) also indicate the sensitivity of the Blackbox Expert to a change in the number of data items available. Using the relationship between result produced and data items available to the Blackbox Expert depicted in Figure 4(a) and Figure 4(b), the developers of the Blackbox Expert are able to set the minimum availability for the data items required, given the desired quality for the result to be produced. Of course, the specific level chosen for the Blackbox Expert, or any other rule-based system, will be dependent upon the environment in which the system must operate.

This method of studying the dynamic properties of a rule-based system by



(a)

GOAL	SEMANTICS
Goal 7	Detect if all balls located
Goal 8	End of game, no desirable beams
Goal 9	End of game, all balls located
Goal 11	Terminate line of reasoning
Goal 25	Mark Location as empty, and it is certain
Goal 133	Place a certain ball, and mark square as empty, and certain
Goal 136	Conflict on placing empty, and conflict on placing ball

(b)

FIGURE 5. Sensitivity of specific goals.

examining goals achieved for each test case can be contrasted with the view provided by simply considering the rules fired when solving each test case. Figure 4(c) presents the ratio of rules fired at each information deficit to the number of rules fired with an information deficit of 0, and the ratio of the number of goals achieved at each information deficit to the number of goals achieved with an information deficit of 0. This gives us the percentage of goals achieved (rules fired) at each information deficit used in the experiment compared to goals achieved (rules fired) with an information deficit of 0. At an information deficit of 0.8, the number of rules fired decreases by 60%, but the number of goals achieved declines by only 39% (using the liberal counting strategy); even though the Blackbox Expert is still able to achieve over 60% of the goals it achieved with an information deficit of 0, it is nearly unable to solve any portion of the Blackbox puzzle (SCORE is 291). Examining goals achieved, rather than simply measuring rules fired, definitely provides us with an in-depth view of the problem-solving activity of the Blackbox Expert.

Impact of specific goals: Let us now consider the impact of achieving specific goals on the result produced by the Blackbox Expert, or the ability of the Blackbox Expert to produce a result. Figure 5(a) shows a sample of goals that were affected by the change in information deficit that was experienced by the Blackbox Expert. For each of the goals, we plot the number of test cases in which the goals were achieved as the data items available were reduced. In Figure 5(a) we consider the number of test cases in which a goal is achieved rather than total number of times a goal is achieved because we only wish to determine if sufficient data items are available for a goal to be achieved; we do not care if the goal is achieved once, or many times—we simply want to know if it is achieved at all.

Figure 5(b) explains the meaning attached by the rule-base designer to each goal. In order to consider the impact of achieving each goal as the data items available to the Blackbox Expert is reduced, we will examine the path that was responsible for that goal being achieved, but in order to be brief we show only the paths for goal 133 and goal 136 (see Figure 6). Examining the paths that achieve each goal, and the data items required by the rules in each path, we interpret the trends shown in Figure 5 as follows.

Goal 136 is achieved in more test cases, indicating that more conflicts are

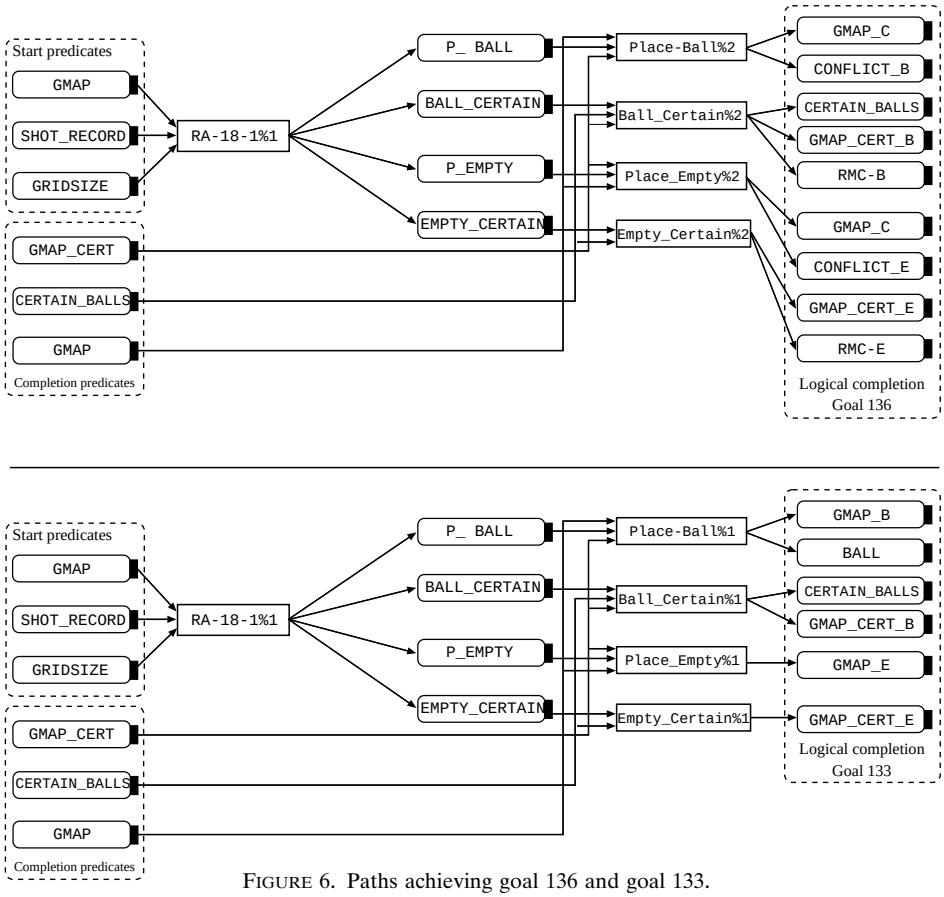


FIGURE 6. Paths achieving goal 136 and goal 133.

generated as the number of data items available are reduced. Figure 6 shows the path responsible for achieving goal 136, and the path responsible for achieving goal 133; goal 133 was the goal achieved by the Blackbox Expert when all data items were available. Considering the paths responsible for the achievement of goal 133 and goal 136, we can conclude that the increased achievement of goal 136 as the number of data items available to the Blackbox Expert are reduced occurs because of an interaction between goal 136 and goal 133. Both the paths achieving goal 136 and goal 133 have the same start rule `RA-18-1%1` and require the same start predicates {`GMAP, SHOT_RECORD, GRIDSIZE`}. As we reduced the data items available, the data items as identified by the start predicates for both paths were still available to the Blackbox Expert, but the data items identified by the completion predicates were not. As a result, all the rules in the path responsible for achieving goal 136 fire, but not all the rules in the path achieving goal 133 can fire; thus, goal 136 is achieved rather than goal 133, signaling a conflict. The conflict signalled by the Blackbox Expert indicates that while the Blackbox Expert had sufficient data items available to recognize a specific configuration in the Blackbox, it was unable to draw a conclusion (updating grid squares) based upon that configuration because it required access to data items that were unavailable.

Goal 7 exhibits a rapid decline; goal 7 can only be achieved if sufficient data items are available for the Blackbox Expert to detect that all the balls in the grid have been located. This is only likely to occur in situations where most of the data items regarding the contents of the grid squares are available. The non-achievement of goal 7 impacts the ability of the Blackbox Expert to detect when an end game situation has been reached. In the case that the Blackbox Expert is unable to detect end game situations, it tends to fire more beams, resulting in an elevated score.

Goals 8 and 9 form a set of goals in which at least one of the members of the set has to be accomplished; problem-solving terminates when either goal 8 or goal 9 is achieved. Of course, problem-solving must terminate, even when the number of data items available is small; thus in each test case, either goal 8 or goal 9 is achieved. As the data items available are reduced, the number of test cases which terminate by achieving goal 9 reduce, causing the number of test cases which terminate by achieving goal 8 to increase.

Goal 11 exhibits a rapid rise followed by a more gentle decline. The trend exhibited by goal 11 occurs due to an interaction between goal 11 and 198. The paths achieving goals 11 and 198 have the same start rules, and thus the same start sets. However, the completion sets for these two paths are different. Goal 11 is achieved when the data items specified in the completion set of the path achieving goal 198 are unavailable, indicating that the current line of reasoning to achieve goal 198 is to be terminated.

Goal 25 is an example of a goal that is fairly sensitive to the availability of data items, and its achievement directly affects the result produced. The achievement of goal 25 is dependent upon data items that indicate the contents of different grid squares, that indicate the certainty of grid squares, and the entry and exit points for the beams that have been fired. As the data items available are reduced, it is less likely that the Blackbox Expert can detect the scenario required to achieve goal 25. The non-achievement of goal 25 indicates that the quality of the result produced by the Blackbox Expert is reduced because the Blackbox Expert is unable to determine the contents of the grid squares.

It is evident that when determining the impact of the availability of data items on the result produced by a rule-based system, not all goals can be considered to be equal in terms of the role they play in the problem-solving process, in terms of the effect they will have on the result produced by the rule-based system, or in terms of their sensitivity to a change in the availability of data items. Certain goals, while central to the problem-solving process (such as goals 7, 8, 9, and 11 for the Blackbox Expert), may have little direct effect on the final result that is produced by the rule-based system, even if they can be achieved more often. Other goals, such as goal 25 for the Blackbox Expert, have a direct impact on the result produced each time they are achieved.

Goals pursued: Let us now consider if the manner in which the Blackbox Expert chose to pursue goals in our experiment is valid, according to our expectations.

The increased achievement of goal 136 instead of goal 133 is problematic, because it represents a waste of resources, and the result produced by the Blackbox Expert is not as good as it is could be. Better results are possible because the contents of several grid squares are known by the Blackbox Expert, but it is not able to update the grid to indicate this. As it is currently designed, the firing of rule `RA-18-1%1`

(see Figure 6) in this situation serves no other purpose, but to generate the conflict, which is not useful. It is clear that `RA-18-1%1` should be redesigned so that it fires only when it can record the contents of the grid squares about which it is able to draw conclusions.

The decline in the ability of the Blackbox Expert to achieve goal 7 is acceptable. Even though this represents a degradation in the ability of the Blackbox Expert to solve Blackbox puzzles, the degradation due to the reduced achievement of goal 7 represents a graceful reduction in the ability of the Blackbox Expert's problem solving ability. However, it should be noted that as the information deficit becomes large, the Blackbox Expert may start to fire too many beams, resulting in an elevated SCORE; this can be problematic.

The achievement of goal 9 indicates that the Blackbox Expert has terminated the solving of a test case because it has determined that the puzzle has been solved, and the achievement of goal 8 indicates that solving of a test case terminated because the Blackbox Expert determined that there were no beams left to fire that were likely to help it in completing the result for a test case. As the number of data items available are reduced, it is part of the specification for the Blackbox Expert to determine that it cannot improve the current result for a given test case by firing more beams, and then decide to terminate its problem solving effort.

The trend in the achievement of goal 11 could be an indication of a problem in the design of the start rules for the paths achieving goal 198. The rules in the start set for the path achieving goal 198 fire, but then the action to be carried out by the other rules in the path achieving goal 198 are aborted, and goal 11 is achieved instead, terminating the line of reasoning with no progress made in the result being produced. This is a waste of resources.

While the achievement of goal 25 is reduced, indicating that the result produced by the Blackbox Expert will have a higher SCORE, the major drop in the number of test cases in which goal 25 is achieved occurs when the information deficit is quite high (information deficit > 0.6) and is as expected. This trend in the achievement of goal 25 should be considered when setting the minimum information deficit under which the Blackbox Expert should operate, as this can become a problem if the information deficit faced by the Blackbox Expert becomes too large.

Summary: By studying the dynamic properties of the Blackbox Expert, we were able to get a more in-depth view and expose a number of potentially problematic phenomena, none of which could be revealed by static validation methods.

- We were able to determine when the absence of specific data items caused a degradation in the Blackbox Expert's ability to solve problems. This reveals which data items are important for problem-solving, in terms of their availability to the system.
- We were able to discover cases in which the data items in the start set of a path were available, but the data items in the completion set were unavailable, leading to wasted problem-solving effort. This lead us to reconsider the conditions used on the LHS of the start rules of the path.
- We were able to validate our expectations with regard to the role of goals in problem-solving, by comparing the trends in achievement of specific goals against their impact upon the result produced. This allowed us to verify that

the Blackbox Expert's ability to solve Blackbox puzzles degrades gracefully as the data items available were reduced.

Modifying the design of future versions of the Blackbox Expert in light of what we have learnt in this demonstration and what we could learn by a complete study of the Blackbox Expert's dynamic properties will give us considerable confidence in the reliability of the system. Our confidence in the system will be much greater than what it would have been if our validation were restricted only to verifying static properties of the system,† and testing the input–output relationship without consideration of the ''inner workings'' of the system.

5. Validation in the life-cycle

The previous section demonstrated the utility of our approach on a specific rule-based system. This section, in contrast, focuses upon the domain-independent aspects of the approach, showing how it can be used in the development of an arbitrary rule-based system. We describe how validation of dynamic properties fits naturally into the life-cycle for rule-based systems. Rather than prescribe a specific life-cycle model, we now examine the aspects of each common life-cycle phase in relation to our method.

Specification and conceptual modelling phases: In contrast with many other approaches, we take a ''minimal'' approach to the specification of rule-based systems. Our method imposes few constraints on the structure and function of the required system. We assume that developers will make a statement-of-requirements early in development, and will create a ''conceptual model'' of the system as knowledge is acquired and analysed. This is in accordance with most currently recommended methodologies, for example (Steels, 1990; Clancey, 1992; Wielinga, Schreiber & Breuker, 1992). We require only that the following information be present in such descriptions, which should not be overly burdensome to specify:

- required classes of input, output and intermediate information;
- required goal states;
- significant relationships required between goals (indicating when a specific goal should be achieved).

Design and implementation phases: The role of the design phase is to decide how the descriptions obtained during the specification and conceptual modelling phase will be realized as an operational system. In the case of an OPS5-like implementation, the design commitments we require are as follows:

- representation of the conceptual classes of information by means of predicates and fact templates;
- identification of the combinations of predicates which correspond to the goal states: these specify our *logical completions*;
- encoding of the tasks to be performed to solve the problem by means of rules.

One of the reasons we chose an OPS5-like language on which to develop our

† Note that the Blackbox Expert had already been subjected to static verification by the CLIPS CRSV utility prior to the the work described in this paper.

approach is that of generality: the CLIPS language includes structured data objects, procedural rule actions (including *if–then* conditional actions), control rules, and calls to external functions. (The Blackbox Expert employs all of these programming techniques.) This permits our method to cope with realistic rule-based systems, not merely ''clean'' rule bases.

Validation and verification phases: Once an implementation exists, Path Hunter can be used to build the structural model of the system. This permits faults to be detected, where there are problems in building the model (for example, where it is impossible to identify any paths for a given logical completion) or where the model includes anomalous structures (for example, apparent irregularities in some paths). Verification is not the topic of this paper, so we will not dwell on this issue here; the topic considered in more detail in (Preece *et al.,* 1994).

Regardless of whether or not validation of the dynamic properties of the system will be performed as described in the previous section, it will always be necessary to test the system on a selection of cases. There are two types of criteria to be met in creating a suitable set of test cases:

Functional criteria: These criteria concern the tasks that the system is required to perform, as for any software system (Adrion, Branstad & Cherniavsky, 1982). Guidance as to the composition of the test set with respect to functional criteria will be provided by the descriptions of tasks, input and output in the specification and conceptual model.

Structural criteria: These criteria concern the exercising of each component of the system—again, as for any software system (Adrion *et al.,* 1982). Here our structural model is especially helpful, since we can ensure that test cases are included which exercise all paths (and, hence, all rules also). This problem is considered in detail in Preece *et al.* (1994).

Taken together, the implemented system, the set of test cases, and the design and specification descriptions provide the inputs to the dynamic validation procedure. It is necessary to decide at this stage whether to validate the system with incomplete data and, if so, how the incompleteness will be simulated. Our information deficit metric is domain-independent for this purpose.

There is always a trade-off to be made between the need to produce a reliable system and the effort involved in validation. Therefore, it is not possible to make a prescription as to how validation of dynamic properties should be done for an arbitrary rule-based system. However, we recommend the following as a practical strategy.

(1) Ensure that the test case set contains sufficient cases to test all critical functions, and a representative range of non-critical functions (Miller, 1990).
(2) Run the set of test cases and gather trace information as described in Section 3; run each case at different levels of data availability if this is required.
(3) Identify cases for which the performance of the system is unacceptable: look at the dynamic behaviour of the system for these cases and identify faulty rules and paths. In addition, it is probably appropriate to sample a number of cases for which the system seems to be performing acceptably and look at the dynamic behaviour to verify that all is as expected.

(4) If the preceding analyses reveal anomalous or erroneous behaviour with respect to specific goals, examine the achievement profiles for these goals across the whole test set and at different levels of data availability (if appropriate).
(5) If appropriate, examine the performance profile for the system as a whole, at different levels of data availability, to identify critical or anomalous features.

Note that this is a *validation* procedure (as opposed to verification) because the system is compared again the *expectations* of the domain experts and would-be users, rather than against a rigid functional specification.

Our structural model plays a valuable role in identifying the causes of any anomalous behaviour that is identified as the rule-based system is validated, because it describes the rule base at a coarser level of granularity than that of individual rules, which is directly related to the specifications and conceptual model.

6. Conclusion

In this paper, we have demonstrated how dynamic properties of a rule-based system can be validated. The dynamic problem-solving behaviour of the system is revealed using a structural model of rule execution paths, which is in turn used to analyse run-time traces. Traces are obtained from running the system on a set of test cases, at different levels of data availability. The rule base designer is thus able:

- to determine when the absence of specific data items causes a degradation in the systems ability to solve problems (hence, which data items are most important for problem-solving);
- to discover wasted problem-solving effort (cases in which the data items in the start set of a path were available, but the data items in the completion set were unavailable);
- to validate the domain experts' expectations with regard to the role of specific goals in problem-solving, and the result produced.

Overall, we have demonstrated (using a complex rule-based system as a test-bed) that the information provided to the rule-base designer by studying dynamic properties enhances the rule-base designer's ability to validate the design of a rule-based system over what can be accomplished with static techniques.

References

ADELMAN, L. (1991). Experiments, quasi-experiments, and case studies; a review of empirical methods for evaluating decision support systems. *IEEE Transactions on Systems, Man, and Cybernetics,* **21,** 293–301.

ADRION, W. R., BRANSTAD, M. A. & CHERNIAVSKY, J. C. (1982). Validation, verification and testing of computer software. *ACM Computing Surveys,* **14,** 159–192.

AYEL, M. & LAURENT, J.-P., Eds (1991). *Verification, Validation and Test of Knowledge-based Systems.* London: John Wiley and Sons.

BATAREKH, A., PREECE, A. D., BENNETT, A. & GROGONO, P. (1991). Specifying an expert system. *Expert Systems with Applications,* **2,** 285–303.

CLANCEY, W. J. (1992). Model construction operators. *Artificial Intelligence,* **53,** 1–115.

GIARRATANO, J. & RILEY, G. (1989). *Expert Systems*: *Principles and Programming.* New York, NY: PWS-Kent.

GINSBERG, A. & WILLIAMSON, K. (1993). Inconsistency and redundancy checking for quasi-first-order-logic knowledge bases. *International Journal of Expert Systems*: *Research and Applications,* **6,** 321–340.

GROSSNER, C. (1994). *Models and tools for cooperating rule-based systems.* PhD thesis, McGill University.

GROSSNER, C., GOKULCHANDER, P., PREECE, A. & RADHAKRISHNAN, T. (1993). Data distribution in organizations of cooperating expert systems. *12th International Workshop on Distributed Artificial Intelligence,* Pittsburgh, PN, USA.

GROSSNER, C., LYONS, J. & RADHAKRISHNAN, T. (1991). Validation of an expert system intended for research in distributed artificial intelligence. *2nd CLIPS Conference, Johnson Space Center,* Houston, TX, USA.

GROSSNER, C., LYONS, J. & RADHAKRISHNAN, T. (1992). Towards a tool for design of cooperating expert systems. *4th International Conference on Tools for Artificial Intelligence,* Arlington, USA.

GROSSNER, C., PREECE, A., GOKULCHANDER, P., RADHAKRISHNAN, T. & SUEN, C. (1993). Exploring the structure of rule based systems. *Proceedings of the 11th National Conference on Artificial Intelligence* (*AAAI 93*), pp. 704–709. Washington D.C.

GUPTA, U. G. (1990). *Validating and Verifying Knowledge-based Systems.* Los Alamitos, CA: IEEE Press.

HAYES-ROTH, F. (1985). Rule-based systems. *Communications of the ACM,* **28,** 921–932.

KIPER, J. D. (1992). Structural testing of rule-based expert systems. *ACM Transactions on Software Engineering and Methodology,* **1,** 168–187.

KIRANI, S., ZUALKERNAN, I. A. & TSAI, W.-T. (1992). *Comparative evaluation of expert system testing methods.* Technical Report TR 92-30, Department of Computer Science, University of Minnesota, MN.

LAFFEY, T. J., COX, P. A., SCHMIDT, J. L., KAO, S. M. & READ, J. Y. (1988). Real-time knowledge-based systems. *AI Magazine,* **9,** 27–45.

MESEGUER, P. & VERDAGUER, A. (1993). Verification of multi-level rule-based expert systems: theory and practice. *International Journal of Expert Systems*: *Research and Applications,* **6,** 163–192.

MILLER, L. (1990). Dynamic testing of knowledge bases using the heuristic testing approach. *Expert Systems with Applications,* **1,** 249–270.

MILLER, L. A., GROUNDWATER, E. & MIRSKY, S. M. (1993). *Survey and assessment of conventional software verification and validation methods.* Technical Report NUREG/CR-6018; EPRI TR-102106; SAIC-91/6660, Science Applications International Corporation; U.S. Nuclear Regulatory Commission; Electric Power Research Institute.

NAZARETH, D. L. (1993). Investigating the applicability of petri nets for rule-based system verification. *IEEE Transactions on Knowledge and Data Engineering,* **4,** 402–415.

O'KEEFE, R. M., BALCI, O. & SMITH, E. P. (1987). Validating expert system performance. *IEEE Expert,* **2,** 81–90.

O'KEEFE, R. M. & O'LEARY, D. E. (1993). Expert system verification and validation: a survey and tutorial. *Artificial Intelligence, Review,* **7,** 3–42.

PREECE, A., GROSSNER, C., GOKULCHANDER, P. & RADHAKRISHNAN, T. (1994). Structural validation of expert systems: Experience using a formal model. In J. LIEBOWITZ, Ed. World Congress on Expert Systems. Macmillan New Media. Published on CD-ROM.

PREECE, A. D., SHINGHAL, R. & BATAREKH, A. (1992). Verifying expert systems: a logical framework and a practical tool. *Expert Systems with Applications,* **3,** 421–436.

PREECE, A. D. & SUEN, C. Y. (1993). Special issue on verification and validation. *International Journal of Expert Systems*: *Research and Applications,* 6(2/3).

RICH, E. (1983). *Artificial Intelligence.* New York, NY: McGraw Hill.

RUSHBY, J. & CROW, J. (1990). *Evaluation of an expert system for fault detection, isolation, and recovery in the manned maneuvering unit.* NASA Contractor Report CR-187466. Menlo Park CA: SRI International.

SIMON, H. A. (1973). The structure of ill-structured problems. *Artificial Intelligence,* **4,** 181–201.

STEELS, L. (1990). Components of expertise. *AI Magazine,* **11,** 29–49.

WIELINGA, B. J., SCHREIBER, A. T. & BREUKER, J. A. (1992). KADS: a modelling approach to knowledge engineering. *Knowledge Acquisition,* **4,** 5–54.

ZLATAREVA, N. & PREECE, A. (1993). An effective logical framework for knowledge-based systems verification. *Proceedings 1993 Florida AI Research Symposium* (*FLAIRS 93*), Orlando, FL, USA.