Technical Report

Department of Computer Science
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 97-018

Verification and Validation of Knowledge-Based Systems

by: W. T. Tsai and
R. Vishnuvajjala

Verification and Validation of Knowledge-Based Systems

W. T. Tsai, R. Vishnuvajjala
Department of Computer Science
University of Minnesota
Minneapolis, MN 55455

D. Zhang
Department of Computer Science
California State University at Sacramento
Sacramento, CA, 95819

ABSTRACT

Knowledge-Based systems are being used in many applications areas where their failures can be costly because of the losses in services, property or even life. To ensure their reliability and dependability, it is important that these systems are verified and validated before they are deployed. This paper provides perspectives on issues and problems that impact the Verification and Validation (V&V) of these systems. Some of reasons why V&V of Knowledge-Based systems is difficult are presented. The paper also provides an overview of different techniques and tools that have been developed for performing V&V activities. Finally, some of the research issues that are relevant for future work in this field are discussed.

1. INTRODUCTION

Expert systems (ES) are intelligent computer systems whose problem-solving performance would rival or exceed human performance in narrow areas of human endeavor, whereas knowledge-based systems (KBS) are computer systems that assist in problem solving. Expert systems and knowledge-based systems are among the fruitful spin-offs from artificial intelligence. The worldwide expert system expenditures in 1993 totaled $595 million. Today, these systems have found their way into many applications such as knowledge publishing, shuttle mission experiment planning, crew scheduling, medical diagnosis, computer configuration, process control, regulation compliance advising, financial advising and nuclear engineering.

As these knowledge systems (KS) (We use KS to denote either ES or KBS.) are deployed in settings where system failures may result in loss of productivity, decision-making, property, business, services, or even life, the reliability and dependability of the systems become important [Barker 89, Gupta 91, Tsai 90b]. It is imperative that the systems are thoroughly verified and validated before they can be deployed in the field. Verification and Validation (V&V) of KS therefore is a key aspect of the development of KS. Many V&V techniques and tools have been proposed, developed and implemented [Ayel 91a, Culbert 90, Ginsberg 93, Gupta 91, Hayes-Roth 94, Lydiard 92, Miller 94, Nazareth 89, O'Keefe 93, O'Leary 94, Plaza 93, Preece 93, Zlaterava 94].

As the use of KS in various areas grows, many commercial KS are being developed. Many techniques have been applied to the development of these KS and experience has been gained about some of the specific issues and problems faced during the development of KS in the industry. These issues can be useful in understanding the progress made and the some of the shortcomings of existing V&V techniques for KS in the context of industrial environments. Table 1 summarizes some surveys about current industrial practice in V&V of KS [Hamilton 91, O'Leary 91].

Survey Subject	Response
performance criteria	75% systems were less accurate than expected; 65% systems were less accurate than the experts; users, more often than developers, estimated the expert systems as being less accurate than expected and less accurate than the expert
requirements definition	52% systems did not have documented requirements; 43% systems used prototypes for requirements; 35% said requirements were hard to develop
development information	40% systems followed lifecycle of cyclic model; 22% systems did not follow any lifecycle model; average of 23 person-months to develop
V&V activities performed	Most V&V activities relied on comparison with expected results and expert checking; 59% had domain expert check Knowledge Base (KB); 24% development effort was spent on V&V; 100% users rated V&V of expert system as hard; 27% developers said V&V were hard; there was wide range of V&V techniques used
V&V issues encountered	test coverage determination (63%); knowledge validation (60%); real-time performance analysis (33%); problem complexity (40%); modularity (27%); certification (11%); understandability (10%); configuration management (20%); validation of Inference Engine (IE) (19%); analysis of certainty factors (86%)

Table 1. Results of A Survey.

From table 1, it is apparent that KS share many characteristics with conventional software. The lack of well-documented requirements, the use of prototypes as requirements and the absence of a well-defined life-cycle model are common to many conventional software development environments also. For instance, inconsistencies, incompleteness, and ambiguity in requirements specifications have been identified as major sources of bugs in software development [Beizer 90]. But KS also have certain components such as the IE, and domain knowledge stored in a KB, which are not present in conventional software. Thus, while many issues that impact the V&V of conventional software such as the lack of requirements documents need to be considered, additional issues such as the validation of the IE and its interactions with the KB also need to be addressed. This paper presents a perspective on the issues related to the V&V of KS and some of their goals.

2. DEVELOPMENT OF KS

The development process for a software product specifies how the development tasks are decomposed into sub-tasks or phases. It also determines what by-products are used and generated during the development process. V&V require checking some properties (such as internal

consistency) of a by-product (such as requirements specification) or a comparison of two by-products (such as a specification and the code) generated during the development process. Hence what constitutes V&V for a software product is closely tied to the software's development processes.

Example development processes include the Waterfall model, Spiral Model, exploratory prototyping, and the Object-Oriented paradigm [Pressman 87]. In the Spiral model of development, the system is developed iteratively in a cyclic fashion, with each cycle consisting of different phases such as the requirements and specifications, design, coding and testing. The system is incrementally developed with changes made in each cycle of the development. The by-products of following such a process include requirement documents, design documents, the code that implements the design, test specifications and test suites. One type of binary comparison between two by-products of such a process is to check the requirements documents (which can be expressed using different techniques including structured models such as data flow diagrams, pseudo-languages such as PDL or formal languages such as VDM) against a design document which specifies the functional decomposition and data structures and algorithms. Similarly, another binary test can be conducted by comparing the design document against the code that implements the design.

General Framework for V&V
1. Determination of goals of V&V
2. Generation of problems (inputs) to be used by a V&V method
3. Generation of expected outcomes (outputs) on the selected problems
4. Application and Evaluation of problems

However, sometimes it is difficult to identify the intermediate by-products in some KS development processes. For instance, table 1 shows that 22% of systems did not follow any lifecycle model. In such cases, the different phases in the development processes are not clearly defined, which makes it difficult to identify by-products of the phases. Table 1 also shows that 52% of systems did not have any documented requirements while 43% systems used prototypes as the requirement specifications. Thus in these development processes intermediate by-products are not available for performing V&V. Verification activities (which compare the products of each phase against the goals set at the start of each phase) are difficult to perform in such cases. Only validation of the system against experts or users can be performed in such cases.

Verification and Validation according to the IEEE Standard Glossary [IEEE 90]
Verification: (1) The process of evaluating a system or component to determine whether the products of a given phase satisfy the conditions imposed at the start of that phase. (2) Formal proof of program correctness.
Validation: The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.
Verification and validation (V&V): The process of determining whether the requirements for a system or component are complete and correct, the products of each development phase fulfill the requirements or conditions imposed by the previous phase, and the final system or component complies with specified requirements.

Another characteristic of KS is that the requirements and specifications for an KS are often dynamically changing. An example of a successful production level KS whose requirements are continuously changing is XCON [Barker 89]. When requirements or other artifacts are dynamically changing it is more difficult to perform V&V on these systems than on systems with static artifacts, since it is important to ensure an artifact's internal consistency whenever it changes before it can be compared with the other by-products of the development process.

Nature of faults in development life-cycle [Kirani 94a]
Faults not detected in early phases become progressively more expensive to rectify in later phases.
A fault-prone region (input region that results in failures) may not be uniformly distributed across the complete input space.
Faults in early life-cycle phases may induce a multitude of faults in the final program.
Inconsistency or incompleteness in any phase may result in inconsistent or missing rules in the final program.

Sometimes, there may not be explicit requirements specifications for KS. For example, it is argued that the dynamically changing KS prototype should be viewed as its own specification within the rapid prototyping paradigm of development. For KS, the availability of an explicit requirement and specification corresponds to having a Type I computational theory [Marr 82] for the task being performed by the KS. Verification in this case is then reduced to showing that a specific implementation is consistent with the computational theory. However, more often than not, an explicit computational theory for an KS does not exist. In the absence of a Type I computational theory, alternative artifacts such as archived test suites, domain experts or literature can be used in lieu of an explicit specification.

3. WHY IS V&V OF KS DIFFICULT?

Many factors cause problems in performing the V&V of KS. Such factors include the different knowledge representations, evolving KBs, the large number of rules in KBs and the costs involved in test case generation, execution and evaluation:

3.1 Knowledge Representation Formalisms

Knowledge in a KB may be represented in terms of different formalisms such as *logic* (predicate or propositional logic), *rules*, *frames*, *cases*, *blackboard*, *generic tasks*, *object-oriented* or *semantic networks*. Sometimes, a single formalism is used, thus resulting in terms such as rule-based KB or frame-based KB, but hybrid formalisms, such as the combination of rules with frames, or rules with cases, have also become popular. Due to the interaction of multiple formalisms, a set of new issues need to be considered for KB verification. For example, it has been demonstrated in that because of the object hierarchy and property inheritance in a hybrid KB of rules and objects, new types of anomalies unique to this hybrid representation will arise such as additional subsumption cases, and semantic conflict cases. KBs expressed in different formalisms or their hybrids are thus prone to different types of anomalies and need to be addressed during the V&V process.

3.2 Evolving and Large KBs

KBs represent knowledge in the application domains. Since the application domains in many cases are still evolving, the KBs change and become increasingly large. A prominent example is the XCON experience where the number of rules has increased from 200 in 1979 to 17000 in 1989 [Barker 89]. XCON also illustrates the dynamically changing subject domain and the complexity and large size of the system. As the KBs evolve at such a pace, it becomes difficult for the developer to perform verification of the KBs to ensure that there are no inconsistencies, incompleteness, and redundancies in the KBs. As applications evolve, the specifications, design, code and test cases change. A change in one of these artifacts can cause ripples in the others. For instance, a change to the system requirements can cause changes to the design, and the code and require new test cases. Configuration control to maintain the versions of these artifacts is an important issue when applications evolve.

Relationships between Error, Fault and Failure

According to the IEEE standard terminology of error, fault and failure, errors represent human mistakes which can result in faults in a system. The execution of a faulty system produces failures [Musa 87]. A fault (or commonly referred to as bug) may cause a system to fail on multiple inputs, but each failure can potentially lead to the discovery of a new fault. Thus, multiple failures of a system do not necessarily imply that the system has multiple faults.

3.3 Characteristics of KBs

KBs have several specific characteristics which give rise to new verification problems. For instance, domain knowledge may be attached with confidence factors and/or temporal operators. In the presence of confidence factors, an otherwise "less-harmful" case of redundancy may lead to serious problems, when each redundant piece of knowledge is utilized multiple times leading to erroneous increases in the level of confidence assigned to associated conclusions. Some of the other characteristics of KBs that impact their V&V include the different knowledge types (sure and heuristic), decomposability of KBs, inclusion vs. exclusion of meta knowledge in a KB, the models of a KB (flat vs. hierarchical), and the monotonicity vs. non-monotonicity of KBs.

3.4 Domain Experts in V&V

Domain experts can be actively involved in the V&V of KS in several phases such as detection of anomalies, analysis of results produced by some automated verifier or to perform necessary corrections to a KB [Rousset 88]. Domain experts are especially needed in those systems that do not have documented requirements or use the system prototypes as the requirements. However, the complexity and size of the KBs often make it hard for domain experts to fully comprehend the KBs. Furthermore, even though experts can be used as oracles to generate expected outputs, they often disagree with each other and cost of the involvement of these domain experts can be an issue.

3.5 KS Anomalies

An *anomaly* refers to patterns of common faults with respect to certain analysis techniques. Different development techniques such as procedural, object-oriented (OO) or logic-based, give rise to different forms of anomalies. For instance, data flow anomalies are those anomalies related to the accessing and updating of data items in a program [Subramanian 94]. Some examples of such anomalies are define-define where a variable is updated twice without an in-between use, or define-kill where a variable where a variable is defined and killed without being used. Such anomalies can occur in paradigms where variables are defined and used. Different V&V techniques are also needed to detect such anomalies within an application. For instance, one example of an anomaly specific to the OO paradigm is the message flow anomaly, which involves two or more methods calling each other repeatedly without performing any useful work. Such anomalies are based on sequencing relationships [Kirani 94b] among the methods of objects in an application.

Certain anomalies are specific to KS. For instance, a rule-based KB may have potential inconsistencies, incompleteness, circularity or redundancies among the rules in the KB. Another example of an anomaly in a KS is the deleterious interactions among good rules, which are indigenous to rule sets which reason under uncertainty. When a group of individually correct rules interact together to give an erroneous final conclusion, then these rules are acting together in a deleterious manner. The cause of such fault stems from the fact that these rules are based on uncertainty.

There has been lack of standardization on classification of KB anomalies. Most of the existing verification techniques adopted their own criteria. The definitions of the anomalies in a KB are largely given through examples. The work reported in [Chang 90] offers a detailed account on verification criteria where three types of criteria, namely structural, logical and semantic criteria were delineated.

3.6 Testing of KS

Testing of KS is an integral part of the V&V process. The general framework for testing consists of the following steps: (1) establishing testing criteria, (2) generating test cases (inputs) and expected outcomes on the selected inputs, (3) applying a test method to exercise the software, and (4) evaluating the test outcomes. Testing in general is a labor-intensive and fault-prone process. The difficulties arise from different testing criteria, large input and output spaces and legal test cases generation. Another important factor that makes testing difficult is the high costs involved in performing these activities:

Testing Criteria: A testing criterion defines the goal for comparing a system against a specification. Several criteria for testing KS such as *testability*, *reliability*, *safety*, *completeness*, *consistency*, *robustness* and *usability*. Different testing criteria will lead to different test cases; for instance some test inputs can be used for testing for reliability as well as safety. However, testing for reliability checks whether the system conforms to its specifications while testing for safety checks whether the system does not cause any harmful effects to its environment. When different criteria exist, appropriate selections must be made for performing the testing.

Difficulties in generation of test case inputs: A legal test input is a test input that is consistent with the input specification in the problem specification. Although a problem specification (which is a description of the problem being solved) [Zualkernan 88] may specify *what* a legal test input is, it often does not state how it can be generated. It is difficult to effectively generate a large number of test inputs when the problem specification does not specify the definition of a legal input. If the problem specification specifies the legality criteria and an oracle (an oracle can be a program or a domain expert) is available it can be used to identify the legal (and illegal) test inputs for positive (and negative) testing. Domain literature can also be used to generate the inputs for a test case. If domain literature is used, there is a danger that one can end up with test inputs that are not representative of the real problems solved by experts in the field. If there is no oracle, it is almost impossible to systematically generate a test input. Table 2 summarizes the problems with the generation of test inputs.

Condition	Difficulties
Problem specification does not specify the legality criteria	Difficult
Problem specification specifies the legality criteria but no oracle is available	Difficult
Problem specification specifies the legality criteria and an oracle is available	Less difficult

Table 2. Difficulties in generating test inputs for black-box testing .

Difficulties in generation of test case outputs: Since V&V of an KS requires a large number of test cases, generation of expected outputs for a legal input can be expensive. Although experts can be used as oracles to generate expected outputs, this can be expensive. Expected outputs can also be generated from other sources such as explicit solution specification (which is an abstraction of an implemented solution), archived test cases, or domain literature. The generation of outputs from these sources, however, will in general be difficult. The difficulties for generating expected outputs are summarized in Table 3.

Condition	Difficulties
Problem specification does not specify the legality criteria	Very Difficult
Problem specification does not but solution specification specifies the legality criteria	Difficult but useless for black-box testing
Both problem and solution specifications specify the legality criteria but no oracle is available	Difficult
Both problem and solution specifications specify the legality criteria and an oracle is available	Less Difficult

Table 3. Difficulties in generating expected test outputs .

Input and output spaces for selection of test cases can be huge: The potential input and output spaces for KS are large and the criteria for picking the appropriate inputs and output are not clear. The sizes of the input space, the output space and the number of possible paths for some KS can be estimated by using the data given in [Buchanan 88] (see table 4). As indicated in Table 4, for systems that use either a selection method (such as MYCIN and INTERNIST) or the construction method (such as XCON, XSEL and XFL), the potential sizes of the input and the output spaces for black-box testing are enormous.

High costs of testing: The cost of a testing method not only depends on the cost of test case generation and the cost of evaluation, but also on the cost of loss, i.e., the cost incurred if a fault is missed. Cost of test case generation consists of cost associated with generation of test inputs and expected outputs. Generating expected outputs can be difficult because testing criteria can be unclear or due to lack of oracles. Consistency and completeness methods have less cost of generation compared with dynamic testing methods (see table 5).

System	# objects	#attributes	size of input space	size of the output space
MYCIN	17	257	3.4×10^{6}	6×10^{6}
INTERNIST	571	4100	4.0×10^{169}	3.1×10^{7}
XCON	94	840	3.5×10^{29}	2.1×10^{276}
XSEL	79	329	5.0×10^{24}	-
XFL	74	252	1.3×10^{23}	-

Table 4. Some Estimates of Parameters of KS [Tsai 93a].

The cost of test case evaluation is often less than the cost of test case generation because a significant portion of test case evaluation can be automated For consistency and completeness testing methods cost associated with running a test program on rule base is less than dynamic testing methods.

Cost of loss is the cost incurred if a testing method fails to identify a fault. This cost can be large if the software has been deployed and legal issues are involved. The cost of loss can be reduced if the software bugs can be found in a timely manner, in the different phases of the development process.

Analysis of costs based on different factors such as the development phases, the formalisms used, the testing techniques applied and the bugs detected will facilitate selection of appropriate techniques for testing a KS. For instance, comparison of the random testing techniques versus the partition testing techniques such as input/output partitioning or path partitioning, or comparison of the costs involved in performing the testing early in the development process (for example, performing completeness checks on requirements) versus performing tests later in the process (such as functional testing on the code) can be useful in determining the techniques that can be effectively applied.

Cost of test case generation	Cost of test case evaluation
Consistency and completeness testing methods	Illegal attributes
Data-flow	Unreferenced attribute
Random	Missing rule
Input partition	Redundancy rule
Output partition	Conflict rule
Dynamic-flow	Subsumption rule
Cause-effect	Data-flow
	Random
	Input partition
	Output partition
	Cause-effect
	Dynamic-flow

Table 5. Testing methods ordered in increasing order of costs from top to bottom [Kirani 94a].

In dynamically changing applications such as the KS, regression testing techniques can be used whenever the system is changed. Regression testing involves running existing test suites on the new system to detect faults. If the domain space is small, then all the test cases can be re-executed. However, if the system domain space is large and a huge number of test cases exist, then it may be expensive to execute all the test cases again when only a small portion of the system may have changed or extended. This requires identification of those test cases which will verify the modified portions of the system. Generalized program slicing techniques [Huang 96] can be used as a mechanism to identify the relevant modified portions of the system. Generalized program slicing techniques use some criteria to slice a given system and return only those portions of the system which satisfy the slicing criteria. By maintaining traceability links between the test suites and the system, only the relevant test cases can be extracted from the existing test cases to perform regression testing on the system.

3.7 Other Issues

Some of the other issues that impact the V&V of KS are as follows:

- Of many V&V criteria, the correctness criterion of a KB has received the most attention in KB verification. Other quality requirements of a KB, such as reliability, maintainability, reusability, understandability have not been adequately addressed.
- There is lack of common criteria for evaluating the performance of KB verification tools and methods, therefore making it hard to compare the effectiveness of the tools. Recently, attempts have been made in [Tsai 90a] to establish a set of standard criteria for evaluating KB verification tools.

4. V&V TECHNIQUES

V&V techniques of conventional software systems have been discussed in [Beizer 90]. Broadly speaking, those techniques can be categorized into two groups: *static methods (analysis)* and *dynamic methods (testing)*. Static methods detect faults by analyzing a complete program, but the program is not executed, while dynamic methods require that a program be executed with regard to test suites.

4.1 Static Methods

Static V&V methods can have different objectives such as detecting completeness and consistency faults and proving the correctness of the programs. Techniques in this category range from informal (reading/reviews, inspections and walkthroughs) to semi-formal checks such as type-checking performed by compilers, to formal techniques (axiomatic mathematical proofs). Some recent attempts include the informal analysis such as the use of checklist approach, the formal analysis such as the assertional approach and the object-oriented specification approach.

4.2 Dynamic Methods

Dynamic V&V methods require the execution of a system through the use of test suites. Test cases can be derived either from a *functional* or *structural* viewpoint. In the functional testing,

also known as the *black-box testing*, a program is treated as a black box. The program's implementation details do not matter as the focus is on the requirement specifications. The structural testing, also known as the *white-box testing*, constructs the test cases based on the implementation details such as the control flow and data flow aspects, source language details, and programming styles. Several useful black-box and white-box testing techniques for KS have been evaluated in [Kirani 94a].

Structural testing: Some of the structural testing techniques include cause-effect graph based testing, dynamic flow testing, data-flow testing and path testing [Beizer 90]. For verifying the quality properties of a KB, structural testing makes use of the information about the internal structure of a KB (e.g., causal relationships among rules). Depending on how the internal structures of a KB are modeled, there are different ways of checking KB for anomalies. These approaches are summarized in Table 6.

Approach	Description	Examples of approaches
Decision Table approach	Check for ambiguity, redundancy or completeness based on conditions and actions of rules organized in a Decision Table.	ESC [Cragun 87]
Logical approach	Conduct some logical operations either directly on a KB or an equivalent set of logic formulas of the KB to derive verification results.	COVADIS [Rousset 88].
Petri net approach	Use Petri nets to models KBs and use analysis techniques based on Petri nets to verify them.	PREPARE [Zhang 94].
Integrity constraint satisfaction approach	Use metaknowledge, the knowledge about domain compiled in a KB, in verifying the completeness and consistency of a KB.	[Lafon 91].
Graph approach	Essential idea is to treat the KB as graph generators and to analyze the graphs produced by the generating functions for certain criteria to pinpoint anomalies in a KB.	[Wilkins 86].
Syntactic inspection approach	Relies on analyzing the syntactic properties of a KB to detect potential errors.	COVER [Preece 93].
Incoherence detection approach	Based on the notion of coherence. Verification of a KB is to detect incoherence in it.	[Ayel 91b].
Incidence matrix approach	Based on representing a rule-based KB in terms of incidence matrices and checking for anomalies through matrix multiplications and comparisons	[Botten 92].

Table 6. A summary of Structural Testing methods.

Functional testing: Functional testing is based on program specification and not on its implementation details. Some of the functional testing methods include random testing and partition testing (input and output). In random testing, test cases are selected randomly. They are efficient at detecting faults at a low cost and are useful when a fault-prone region is uniformly distributed across the input space. Partition testing involves selecting test cases from partitions of

input or output spaces. Test cases based on partitions are effective when a program has a non-uniform fault prone region. In case of KS, their specific characteristics need to be considered for performing functional testing. Functional testing of a KB can be based on the specification of a functional description of a KB. When provided with some initial condition (e.g., aggregations of initial facts), a KB will respond accordingly to produce some results (e.g., aggregations of derived facts). Functional testing methods can be used to detect anomalies from aberrations of such predefined input-output patterns. Table 7 summarizes some of the functional testing methods that have been developed for KS.

Approach	Description	Examples of approaches
Machine Learning approach	The essential idea of machine learning based approach is to generate examples from the given KB by using some learning strategies; and confirm the examples to verify its correctness.	[De Raedt 91].
Assumption-based truth maintenance system (ATMS) approach	This approach stems from the idea of truth maintenance system of de Kleer.	[Ginsberg 88].
Relational approach	Based on the concepts of attribute space (which represents a union of domain of all attributes used in rules of a KB) and defining rules as functions on the attribute space. Verifying the KB amounts to detecting certain relations between the rule functions.	[Marathe 89].
Refinement approach	This approach identifies potential errors in a KB through a case database (which represents a set of cases with known conclusions), statistical concepts and heuristics [Ginsberg 85]. It also provides suggestions for appropriate rule modification.	[Ginsberg 85].

Table 7. A summary of Functional Testing Methods.

5. TOOLS FOR KS V&V

In this section we summarize the features of some of the tools that have been developed for V&V of KS. The TEIRESIAS program was the first attempt to automate the rule base debugging process. It was designed as part of the debugging facilities of MYCIN, the infectious blood disease consultation system. By using TEIRESIAS as a tool, the expert could judge whether or not MYCIN's diagnosis is correct, track down the faults in the knowledge base that resulted in an incorrect conclusion, and alter, add, or delete rules to fix errors.

The ONCOCIN rule checker is a knowledge base verification program for ONCOCIN, a KS for oncology protocol management. This program is different from the TEIRESIAS program in that rules were examined as they were entered into the system. The format of the rules consists of action parameter, context, condition, action, and classification. To check for inconsistencies and incompleteness, a table is made which consists of all possible combinations of condition parameter values and their corresponding action (conclusion) parameter values. Conflict, redundancy, subsumption, and missing rules are detected by examining the table.

	CLINT	COVADIS	COVER	IN-DEPTH	KRUST	MELODIA	PREPARE	SACCO
language/ environment.	Prolog Sun workst.	Lisp Sun workst.	Prolog & C Sun-4/300	Comm. Lisp Sun-4/260	Sun-3/280	Pascal IBM 3090	C DEC 3100	SACKOOL
knowledge representation	predicate logic	production rules	predicate logic	predicate logic	frames	proposit. logic	predicate logic	predicate logic
KB model	flat	flat	flat	hierarchical	flat	flat	flat	flat
domain dependency	depend.	depend.	independ.	independ.	depend.	depend.	independ.	depend.
approach	machine learning	logical	syntactic inspection	KB-reduction	refinement	logical	petri net & pattern recog.	incoherence detection
dependence of IE	no	yes	no	no	no	no	no	no
participation of expert in detection	yes	yes	no	no	no	no	no	no
handling of certainty factors	no	no	no	yes	no	no	no	no
use of heuristics in detection	no	yes	yes	no	no	no	yes	yes
detection (d) vs. detection & correction (dc)	dc	d only	d only	d only	dc	d only	d only	dc
types of anomalies detectable	inconsist.	inconsist.	redundancy, ambivalence, circularity, deficiency	inconsist., redundancy, circularity, useless KB objects	inconsist.	inconsist., redundancy, hidden theorems	inconsist., redundancy, circularity, incomplete.	incoherence
size of KB tested	N/A	up to 200 rules	up to 550 rules	up to 334 rules	up to 200 rules	up to 10562 rules	up to 500 rules	N/A
computation time	N/A	N/A	exponential worst case 550 rules < 3.5 hours	exponential worst case 334 rules < 20 hours	200 rules < 31 minutes	10562 rules < 6 minutes	exponential worst case 500 rules < 1 minute	N/A
reference	[De Raedt 91]	[Rousset 88]	[Preece 93]	[Meseguer 93]	[Craw 91]	[Charles 91]	[Zhang 93]	[Ayel 91b]

Table 8. Features of some KB verifiers.

The CHECK is a rule base verification program [Nguyen 87] for LES, the Lockheed ES shell. It is claimed to be an extension of the ONCOCIN 's rule checker. It differs from the ONCOCIN 's rule checker in that CHECK is applied to the entire set of rules rather than just subsets of the

rules. In addition to the types of faults the ONCOCIN's rule checker detects, CHECK includes the detection of several extra types of faults, such as circular rules, unnecessary if conditions, and unreachable conclusions.

EVA (Expert systems Validation Associate) is an integrated set of tools for validating KS [Chang 90]. Its goal is to include all the necessary generic tools for validation of any KS developed in any expert system shell for any problem domain. Several tools such as structure checker and logic checker have been designed to detect anomalies of a particular nature.

KB-REDUCER, a verifier based on KB-reduction approach, is capable of detecting inconsistencies and redundancies. It has been used to analyze several KB of up to 370 rules in size. The worst case time complexity in generating environments may be exponential.

A list of some recently developed tools is shown in Table 8, along with their features. In addition to the effort made by the researchers, vendors have developed and integrated V&V tools and utilities for their own expert system shells (e.g., CRSV for CLIPS and Automatic Validation function for EXSYS).

6. RESEARCH ISSUES

There is a spectrum of research issues in KS V&V, some currently being studied, some yet to be investigated.

More experimental data on effectiveness of different KS verification approaches are needed: Comparative study is needed on existing KS V&V approaches using veridical systems to determine what approach is effective for detecting which type(s) of anomalies as well as for determining the costs involved and the scalability of the techniques. Veridical systems represent systems that are deployed in the field and provide realistic data about the size and complexity of systems in the field. Research using veridical systems can help find real issues in V&V of KS. Usefulness of any technique will be validated by their application to veridical systems.

Effectiveness under different contexts: Experiments need to be performed to determine the strengths and weaknesses of different techniques used under different contexts. For instance, experiments needs to be performed to determine which technique would be most effective for rule-based systems versus frame-based systems, or which technique can be used for systems developed using the spiral development paradigm versus the OO paradigms. Some of the other issues that need to be considered are the lifecycle phases in which a technique is effective, the representation formalisms and the cost-effectiveness of the techniques.

One technique called life-cycle mutation testing (LCMT) can be useful for evaluating testing methods for a KS at each phase of the development process [Kirani 94a]. In this technique, intentional faults called mutants are introduced one at a time into a program, and a testing method to be evaluated is applied to the program. If the testing method fails to identify the failures resulting from a mutation, the mutant is said to be live. Otherwise, the mutant is considered killed. A test method's adequacy is determined by the number of mutants it is able to kill. In LCMT, a mutant is introduced in all by-products of development such as problem specification, solution specification, high-level design and the implementation code. The LCMT study was conducted on a diagnostic VLSI manufacturing ES called MAPS. MAPS is based on

authentic reasoning methods of an expert with 12 years of experience and it performed better than another expert with seven years of experience. By injecting different faults at different phases of the development process, the study performed an analysis of the performance and cost-effectiveness of various testing methods in different phases of the development process. For instance, the study found that black-box testing methods killed more mutants in initial life-cycle phases compared with the implementation phase. Further studies focusing on the effectiveness of verification approaches considering other factors such as the knowledge representations and the size of the KBs are needed.

Evaluation of costs of V&V methods: Issues regarding V&V costs need to be researched further. Costs associated with V&V depend on many factors. For instance, it is not clear what the relative costs will be in detecting or failing to detect a fault using a static technique (such as inspections) versus using a dynamic technique (such as some black-box testing). Such an analysis can be useful in determining which technique should be used under various circumstances.

Scaling up of techniques and tools. Many tools and techniques have been mainly tested on KBs that are fairly small in size (e.g., hundreds or thousands rules in a rule-based KB). More data are needed about the effectiveness and efficiency of these tools when applied to large KBs. Systems with large KBs are not far fetched and many have already been on the horizon.

Focus on entire system, not just the KB: Many existing V&V tools deal mainly with the correctness aspect of a KB, but there are many other components in a KS, such as the IE, the explanation module, the knowledge acquisition module, the communication module and various other interfaces. A fault in any of these components can potentially lead to a failure in the KS. Therefore, support is needed for V&V of each of these components as well as the entire system.

Incorporate V&V into development processes: The focus on most existing KS V&V approaches is to detect and correct all types of anomalies at the end of KS development process, i.e., the coding stage. Little effort is made to reduce the chances of introducing anomalies into KS at earlier phases. For instance, techniques that test the completeness and consistency of software specifications [Tsai 93b] can be used to detect anomalies in the specification phase. V&V techniques need to be integrated with the KS development processes.

Standards. There is a need for standardizing terminology and definitions of anomalies.

Theoretic foundations. We need to establish some formal semantics for KB anomalies. There has been effort in this direction [Rushby 88, Rushby 89].

Agent-based applications: The growth of Internet and the World Wide Web has led to an explosion in computer-based tasks and services, and the technology of *software agents*. Agents are computer programs that make autonomous decisions based on the data received from agencies and support common tasks such as filtering of news and mail, scheduling of meetings, and selection of music. As the popularity of agent-based applications grows, verification of their behavior becomes critical to avoid failures. Research into V&V techniques needs to address the special characteristics of agent-based applications, such as testing techniques for programs involving large number of distributed autonomous agents.

Software reliability models: Software reliability has been a subject for research for conventional software for many years, however, software reliability models have not received

great attention yet [Bastani 90]. Software reliability models can be used to estimate mean time between software failures and is essential for safety-critical applications such as nuclear reactor control, where failures can be costly leading to loss of property or life. These models can indicate when the V&V activities have been adequately performed.

ACKNOWLEDGEMENTS

We would like to express our sincere thanks to Dr. Imran Zualkernan and Dr. Shekhar Kirani who have collaborated with us for several years in related areas.

REFERENCES

[Ayel 91a] M. Ayel and J.P. Laurent (Eds.), *Validation, Verification and Test of Knowledge-Based Systems*, John Wiley & Sons, Chichester, 1991.

[Ayel 91b] M. Ayel and J.P. Laurent, "SACCO-SYCOJET: Two Different Ways of Verifying Knowledge-Based Systems", In *Validation, Verification and Test of Knowledge-Based Systems*, M. Ayel and J.P. Laurent (Eds.), John Wiley & Sons, Chichester, 1991, pp. 63-76.

[Barker 89] V.E. Barker, D.E. O'Connor, J. Bachant and E. Soloway, "Expert Systems for Configuration at Digital: XCON and Beyond", *Communications of ACM*, Vol. 32, No. 3, 1989, pp. 298-318.

[Bastani 90] F. B. Bastani and I. R. Chen, "Assessment of Reliability of AI Programs", in Proc. of IEEE Conference on Tools for AI, 1990, pp. 753-759.

[Beizer 90] B. Beizer, *Software Testing Techniques*, Van Nostrand Reinhold, New York, 1990.

[Botten 92] N. Botten, "Complex Knowledge Base Verification Using Matrices", In Lecture Notes in Artificial Intelligence, F. Belli and F.J. Radermacher (Eds.), Springer-Verlag, Berlin, 1992, pp. 225-235.

[Buchanan 88] B.G. Buchanan, "AI as an experimental science", In *Aspects of Artificial Intelligence*, J.E.Fetzer (Ed.), Kluwer Academic Publishing, 1988, pp. 209-250.

[Chang 90] C.L. Chang, J.B. Combs and R.A. Stachowitz, "A Report on the Expert Systems Validation Associate (EVA)", *Expert Systems with Applications*, Vol. 1, 1990, pp. 217-230.

[Charles 91] E. Charles and O. Dubois, "MELODIA: Logical Methods for Checking Knowledge Bases", In *Validation, Verification and Test of Knowledge-Based Systems*, M. Ayel and J.P. Laurent (Eds.), John Wiley & Sons, Chichester, 1991, pp. 95-105.

[Cragun 87] B.J. Cragun and H.J. Steudel, "A Decision-Table-Based Processor for Checking Completeness and Consistency in Rule-Based Expert Systems", *International Journal of Man-Machine Studies*, Vol. 26, 1987, pp. 633-648.

[Craw 91] S. Craw, "Judging Knowledge Base Quality", In *Validation, Verification and Test of Knowledge-Based Systems*, M. Ayel and J.P. Laurent (Eds.), John Wiley & Sons, Chichester, 1991, pp. 207-219.

[Culbert 90] C. Culbert (Ed.), *Special Issue: Verification and Validation of Knowledge-Based Systems, Expert Systems with Applications*, Vol. 1, No. 3, 1990.

[De Raedt 91] L. De Raedt, G. Sablon and M. Bruynooghe, "Using Interactive Concept Learning for Knowledge-base Validation and Verification", In *Validation, Verification and Test of Knowledge-Based Systems*, M. Ayel and J.P. Laurent (Eds.), John Wiley & Sons, Chichester, 1991, pp. 177-190.

[Ginsberg 85] A. Ginsberg, S. Weiss and P. Politakis, "SEEK2: A Generalized Approach to Automatic Knowledge Base Refinement", In Proc. of International Joint Conference on AI, 1985, pp. 367-374.

[Ginsberg 88] A. Ginsberg, "Knowledge-Base Reduction: A New Approach to Checking Knowledge Bases for Inconsistency and Redundancy", In Proc. of Seventh National Conference on AI, 1988, pp. 585-589.

[Ginsberg 93] A. Ginsberg and K. Williamson, "Inconsistency and Redundancy Checking for Quasi-First-Order-Logic Knowledge Bases", *International Journal of Expert Systems*, Vol. 6, No. 3, 1993, pp. 321-340.

[Gupta 91] U.G. Gupta (Ed.), *Validating and Verifying Knowledge-Based Systems*, IEEE Computer Society Press, Los Alamitos CA, 1991.

[Hamilton 91] D. Hamilton, K. Kelley and C. Culbert, "State-of-the-Practice in Knowledge-Based System Verification and Validation", *Expert Systems with Applications*, Vol. 3, 1991, pp. 403-410.

[Hayes-Roth 94] F. Hayes-Roth and N. Jacobstein, "The State of Knowledge-Based Systems", *Communications of ACM*, Vol. 37, No. 3, 1994, pp. 27-39.

[Huang 96] H. Huang, W.T.Tsai, and S. Subramanian, "Generalized Program Slicing for Software Maintenance", In Proc. Of Software Engineering and Knowledge Engineering, 1996, pp. 261-268.

[IEEE 90] IEEE Standard 610.12-1990, *IEEE Glossary of Software Engineering Terminology,* 1990.

[Kirani 94a] S. Kirani, I.A. Zaulkernan and W.T. Tsai, "Evaluation of Expert System Testing Methods", *Communications of the ACM,* Vol. 37, No. 11, 1994, pp. 71-81.

[Kirani 94b] S. Kirani and W.T. Tsai, "Specification and Verification of Object-Oriented Programs", Technical Report, Computer Science Department, 1994.

[Lafon 91] P. Lafon, "A Descriptive Model of Predicates for Verifying Production Systems", in V*alidation, Verification and Test of Knowledge-Based Systems*, M. Ayel and J.P. Laurent (Eds.), John Wiley & Sons, Chichester, 1991, pp. 149-162.

[Lydiard 92] T.J. Lydiard, "Overview of Current Practice and Research Initiatives for the Verification and Validation of KBS", *The Knowledge Engineering Review*, Vol. 7, No. 2, 1992, pp. 101-113.

[Marathe 89] H. Marathe, T.K. Ma and C.C. Liu, "An Algorithm for Identification of Relations among Rules", In Proc. of IEEE International Workshop on Tools for AI, 1989, pp. 360-366.

[Marr 82] D. Marr, *Vision*, W.H. Freeman, N.Y., N.Y., 1982.

[Meseguer 93] P. Meseguer and A. Verdaguer, "Verification of Multi-Level Rule-Based Expert Systems: Theory and Practice", *International Journal of Expert Systems*, Vol. 6, No. 2, 1993, pp. 163-192.

[Miller 94] L.A. Miller, "Recommended Guidelines for V&V of Various Kinds of Systems at Various Lifecycle Phases", In Proc. Of AAAI-94 Workshop on Validation and Verification of Knowledge-Based Systems, 1994, pp. 1-9.

[Musa 87] J. Musa, A. Iannino and K. Okumoto, *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill, New-York, 1987.

[Nazareth 89] D.L. Nazareth, "Issues in the Verification of Knowledge in Rule-Based Systems", *International Journal of Man-Machine Studies*, Vol. 30, 1989, pp. 255-271.

[Nguyen 87] T.A. Nguyen, W.A. Perkins, T.J. Laffey and D. Pecora, "Knowledge Base Verification", *AI Magazine*, Vol. 8, 1987, pp. 69-75.

[O'Keefe 93] R.M. O'Keefe and D.E. O'Leary, "Expert System Verification and Validation: A Survey and Tutorial", *Artificial Intelligence Review*, Vol. 7, 1993, pp. 3-42.

[O'Leary 91] D.E. O'Leary, "Design, Development and Validation of Expert Systems: A Survey of Developers", In *Validation, Verification and Test of Knowledge-Based Systems*, M. Ayel and J.P. Laurent (Eds.), John Wiley & Sons, Chichester, 1991, pp. 3-19.

[O'Leary 94] D.E. O'Leary (Ed.), "Special Issue: Verification and Validation of Intelligent Systems: Five Years of AAAI Workshops", *International Journal of Intelligent Systems*, Vol. 9, No. 8-9, 1994.

[Plaza 93] E. Plaza (Ed.), "Validation and Verification of Knowledge-based Systems", *IEEE Expert*, Vol. 8, 1993, pp. 45-81.

[Preece 92] A. Preece, R. Shinghal and A. Batarekh, "Verifying Expert Systems: A Logical Framework and A Practical Tool", *Expert Systems with Applications,* Vol. 5, No. 2-3, 1992, pp. 421-436.

[Preece 93] A. Preece and C. Suen (Eds.), *Special Issues: Verification and Validation of Knowledge-Based Systems*, *International Journal of Expert Systems*, Vol. 6, No. 2-3, 1993.

[Pressman 87] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, McGraw-Hill, New York, 1987.

[Rousset 88] M.C. Rousset, "On the Consistency of Knowledge Bases: the COVADIS System", In Proc. of Eighth European Conference on AI, 1988, pp. 79-84.

[Rushby 88] J. Rushby, "Quality Measures and Assurance for AI Software", *NASA Contractor Report 4187*, October 1988.

[Rushby 89] J. Rushby, "Formal Verification of AI Software", *NASA Contractor Report 181827*, February 1989.

[Subramanian 94] S. Subramanian, W.T.Tsai and S.Kirani, "Hierarchical Data Flow Analysis for OO programs", *Journal of Object-Oriented Programming*, Vol. 7, No. 2, 1994, pp. 36-46.

[Tsai 90a] W. T. Tsai and I.A. Zualkernan, "Towards a Unified Framework for Testing Expert Systems", In Proc. of International Conference on Software Engineering and Knowledge Engineering, 1990.

[Tsai 90b] W. T. Tsai, K. Heisler, D. Volovik, and I. A. Zualkernan, "AI and Software Engineering: A clash of cultures?", In *Computers for Artificial Intelligence Processing*, B. W. Wah and C. V. Ramamoorthy (Eds.), John Wiley and Sons, N. Y., 1990.

[Tsai 93a] W.T. Tsai, I.A. Zualkernan and S. Kirani, "Pragmatic Testing Methods for Expert Systems", *International Journal of AI Tools*, Vol.2, No.2, 1993, pp.181-217.

[Tsai 93b] W. T. Tsai, W. Xie, I. A. Zualkernan, and S. K. Musukula, "A Framework for Systematic Testing of Software Specifications", In Proc, of International Conference on Software Engineering and Knowledge Engineering, 1993, pp. 380-387.

[Wilkins 86] D.C. Wilkins and B.G. Buchanan, "On Debugging Rule Sets When Reasoning under Uncertainty", In Proc. of Fourth National Conference on AI, 1986, pp. 448-454.

[Zhang 93] D. Zhang and D. Nguyen, "A Tool for Knowledge Base Verification", In *Advanced Series on Artificial Intelligence*, Vol.2: *Knowledge Engineering Shells-Systems and Techniques*, N. Bourbakis (Ed.), World Scientific Publishers, 1993, pp. 455-486.

[Zhang 94] D. Zhang and D. Nguyen, "PREPARE: A Tool for Knowledge Base Verification", *IEEE Transactions on Knowledge and Data Engineering,* Vol. 6, No. 6, 1994, pp. 983-989.

[Zlatereva 94] N. Zlatareva and A. Preece, "State of the Art in Automated Validation of Knowledge-Based Systems", *Expert Systems with Applications*, Vol. 7, No. 2, 1994, pp. 151-167.

[Zualkernan 88] I. A. Zualkernan, W. T. Tsai, P. E. Johnson, J. H. Moller, "Utility of Knowledge-Level Specifications", In Proc. of 4th Annual Artificial Intelligence & Advanced Computer Technology Conference, 1988, pp. 79-85.