KNOWLEDGE VERIFICATION

Originally, knowledge-based systems existed as small laboratory prototypes; they were easy to handle and the developers had no difficulty managing their structure and behavior. During recent years, knowledge-based systems have grown from laboratory prototypes to large and complex applications for use under real-life conditions. With the increase in complexity and problem solving capabilities (1), the systems became harder to understand, and all the negative phenomena of the software life-cycle were encountered. Today's knowledge-based systems require strategies for correctness testing just as "conventional" software systems do. However, during the verification of knowledge-based systems, difficulties beyond the verification problems of conventional software development arise.

- The development of knowledge-based systems often starts with vague requirements so that it becomes difficult to determine the system's tasks and whether it performs them correctly. Requirement specifications are often nonexistent, imprecise, or rapidly changing.
- Whereas individual rules are often unstructured, the rules of a knowledge-based system are heavily interdependent. This makes it difficult to determine the execution sequence from a static examination of the knowledge base.
- The logical relationships between data structures are quite complex. With the added effects of rules and

demons (control structures), it is very difficult to maintain an understanding of the system's functionality.
- Development of knowledge-based systems by teams can easily lead to contradictions, redundancies, and missing information in the knowledge base. Similar problems arise when attempting to enhance or modify the knowledge-based system's performance through the addition of rules.

Many knowledge-based systems have been developed to deal with problems in industrial, scientific, and financial applications. Despite the great potential of these systems and millions of dollars invested on their research and development, the major concern faced in industry today is whether these systems are dependable (2,3). Dependability includes such notions as reliability, safety, security, maintainability, and portability (4,5). Reliability is normally defined as the probability that a system will perform correctly according to users' specifications under certain environment conditions for a specified period of time. Safety is related to the probability that hazards don't occur during the execution of a system. Security and safety are closely related, but security is more concerned with the threats to privacy or national security (6). Maintainability defines the degree of difficulty to correct errors in an intelligent system. Portability is concerned with the ease in transferring an Artificial Intelligence (AI) system to a different machine.

To achieve dependability of AI systems, various techniques and tools have been developed (mainly to support the development of correct AI systems). They either try to gain a system that can correctly replay sets of test cases, or they check an existing knowlege base for internal correctness. The approaches differ vastly in their underlying concepts of correctness (e.g., absence of certain phenomena, correctness with respect to test cases, or even logical correctness). In this article, we will discuss the research issues and various techniques in this area.

The article is organized as follows. The first part describes current issues and approaches concerning the certification and evaluation of real-time safety-critical intelligent systems. The next part offers a sketch of our verification technique that is based on the first order logic as its knowledge representation formalism. The final part contains concluding remarks, followed by the bibliography.

CURRENT ISSUES IN VERIFICATION OF KNOWLEDGE-BASED SYSTEMS

In order to identify and understand the problems encountered in the verification of knowledge-based systems, we have to first look at those issues involved in knowledge-base verification that are not normally encountered in the verification of conventional software systems. This will also help us in evaluating various verification techniques and forming a basis for comparing these techniques.

Current Issues

The verification of a knowledge-based system includes ensuring the quality properties of (1) the knowledge base (correctness, robustness, maintainability); (2) the inference engine (soundness and completeness of the inference method, and completeness and correctness of the control strategy); (3) other system components (meta knowledge bases, interfaces, explanation module, knowledge acquisition module, and communication module); and (4) interactions of system components. Next, we explain the various issues involved in a knowledge base verification.

The following sections cover issues related to representation domain and the like.

Knowledge Representation Dependency. Ideally, knowledge base verification techniques should be knowledge representation independent so that they can be applied to systems based on different knowledge representation formalisms. However, most of the existing verification techniques are based on either particular knowledge representation formalisms such as production rules or quasi-first-order formulas, or specific inference methods. Thus each technique has its applicability limit. Though it is desirable to have verification methods which can be applied to different knowledge representation formalisms and different inference methods, this is nevertheless not an easy job since the notion of the correctness of a knowledge base heavily depends on the knowledge representation formalism and inference methods. One attempt to achieve this goal can be found in EVA (Expert Systems Validation Associate) (7). The ultimate goal of EVA was to support verification for knowledge-based systems based on different knowledge representation formalisms. In Ref. 8, a generic knowledge representation formalism is proposed so that systems based on different knowledge representation formalisms can be transformed into the generic formalism. This approach represents an attempt to accomplish representation independent verification.

Domain Dependency. Knowledge base verifiers can be classified into domain dependent and domain independent classes. With a domain dependent verifier, metaknowledge about the problem domain is utilized to perform the verification. A verifier of this kind can be well fit for a particular problem domain. But for a different problem domain it may be totally useless. On the other hand, a domain independent verifier can be applied to many problem domains, but due to its generality and lack of domain specific metaknowledge, it may not be able to do satisfactory work.

Flat Model Versus Hierarchical Model of Knowledge Base. A knowledge base can be considered as a flat structure which contains domain knowledge only or it can be considered as a hierarchical structure which contains both domain knowledge and control knowledge. Most of the verifiers which have been developed so far are based on the flat model assumption. But recently, it has been argued in the work (9) that a hierarchical model is more desired for knowledge base verification than a flat model.

Certainty Factors and Temporal Operators. When confidence factors and temporal operators are present in a knowledge base, it gives rise to some new problems in knowledge base verification. For example, when confidence factors are attached to facts and rules, a redundant rule may cause program errors because a small confidence level may be accumulated into a significant level due to multiple firings of the same rule. Another case involving confidence factors is that conflicting rules, which are regarded as an obvious error in a knowledge base without confidence factors, are no longer considered as an error in a knowledge base with confidence factors.

Monotonicity of Knowledge Base. Nonmonotonic logic has been proposed as an extension of traditional monotonic logic. However, since most of the expert systems are based on a monotonic knowledge base, the existing verification techniques can only deal with a monotonic knowledge base. With the advent of systems such as Operating System 5 that made it possible to develop a nonmonotonic knowledge base, it is now necessary to come to grips with the issue of nonmonotonic knowledge base verification. Some results can be found in Refs. 8, 10, and 11.

The following sections cover the goals and criteria of knowledge base validation.

Verification Criteria. One of the problems with knowledge base verification is that there are no common verification criteria. Different techniques adopt different properties to identify anomalies or errors in a knowledge base. In many cases, the definitions of anomalies or errors used in those techniques are not rigorous and are given through examples. An example of how to discuss verification criteria formally is given in Ref. 7.

Performance Measures. The issue here pertains to the quality of a verification tool (17). For instance, the accuracy of a verifier could be evaluated with regard to the following three aspects: (1) mistaken cases in which correct rules are identified by the verifier as errors; (2) missing cases in which erroneous rules are overlooked by the verifier; (3) the percentage of rules being checked by the verifier.

The following sections cover the role of domain expert.

Participation of Domain Expert in Detection. As mentioned earlier, most of the current verifiers work in a detection-confirmation style: a verifier finds potential errors in a knowledge base and the domain expert confirms the result. Typically, the domain expert is not required to be involved in the detection stage. However, in a machine learning approach (13), a domain expert is required to take part in the first stage in order to facilitate the detection process.

Mode of Operation. Three questions should be answered here. First, when can a verifier be invoked in the rule-based system development cycle? Can it be invoked at every stage or only at the end stage? Finally, who will be in charge of the verification process, the knowledge engineer, the domain expert or a third party or any combination of them?

Detection-Only Versus Detection-and-Correction. Most of the work which has been done so far on knowledge base verification is based on detection-only style: a verifier is used to detect potential errors in a knowledge base and then the domain expert must decide on what to do for the next step. Traditionally, if an error is confirmed by a domain expert, he or she can do one of the following things (12): (1) make the conditions of the problematic rules more specific or more general, (2) make the conclusions of the problematic rules more specific or more general, (3) delete the problematic rules, (4) add new rules to counteract the effects of the problematic rules, (5) modify the confidence factors of the problematic rules. Ideally, we would like to make the process of knowledge base verification less dependent on the domain expert. For example, in Ref. 13 an attempt was made to perform an automatic correction without much of the domain expert's effort.

The following sections cover general issues relating to knowledge-based systems.

Exhaustive Versus Heuristic Checking. The state-of-the-practice in checking for anomalous rules in a knowledge base is through the use of an exhaustive search. However, since this kind of search is time-consuming, a heuristic approach was suggested to do the search (14,15). Basically, the challenge for this issue is how to transform a knowledge base verification problem into a heuristic search problem.

Static Versus Dynamic Checking. Generally speaking, a knowledge base can be divided into its rule base and fact base, with knowledge base verification usually performed on the rule base statically. However, when knowledge base verifications are performed on both the rule base and the fact base, it is possible to do dynamic or static checking depending on whether an inference engine is involved in the verification process. Ideally, dynamic checking is more desirable. A verifier of this type is reported in Ref. 16, which assumes the forward chaining strategy for knowledge base verification. Another example is given in Ref. 8, in which dynamic verification is used to verify a requirement specification language.

Current Approaches

There are a number of approaches that were developed for verification of knowledge-based systems. The extent to which an approach is suited for verifying the desired properties of a knowledge-based system depends on the formalism underlying the representation of the system. Here we discuss some of the widely used approaches in the verification of knowledge-based systems.

Decision Table Approach. In this approach, structural information about a knowledge base is captured in decision tables and algorithms are provided to determine the presence or absence of abnormal properties.

ONCOCIN (18) is a development tool for medical therapy selection based on the formalisms of MYCIN (rules, contexts). After every change or insertion of a rule into the knowledge base the system checks for conflicts, redundancies, and missing rules (a situation exists in which a particular inference is required but no rule succeeds). A table is constructed representing all combinations of variables in the preconditions of the rules together with the values assigned to the variables in actions. This table is algorithmically checked for conflicts, redundancies, subsumptions, and missing rules. ONCOCIN assumes that every combination of variables and values needs a rule. As a result, certain reported missing rules by the system may correspond to meanless combinations.

CHECK (19) can be used to work with knowledge base containing certainty factors. The system supports a backward chaining interpreter that tries to prove some specified goal clauses. In a table, rules are compared with each other to detect dependencies between the preconditions and conclusions of each rule, as well as dependencies between rules and goal clauses. Algorithms are also included to check for cycles, missing rules (every attribute-value combination must be covered), unreachable clauses, and dead-end clauses.

The Expert Systems Validation Associate (EVA) (7) is a wide-range rule base checking system containing algorithms that can be applied to knowledge base developed with different shells such as ART, CLIPS, OPS5, or KEE. It checks rule bases extended by metaknowledge pertaining to constraints and object structures.

Apart from the standard algorithms for table comparisons, EVA relies on a wide range of metaknowledge for judging potential sources of conflicts and incompleteness. By specifying metaknowledge as constraints, the user can transfer semantic information about the application domain to the system. The user defines relations among predicates in terms of metapredicates such as *synonymous, incompatible, inverse, transitive,* or *reflexive.* Similarly, relationships between objects and classes of objects may be defined.

COVADIS (16) is applicable to rule-based knowledge base. Contraints are used for specifying contradictions. COVADIS assumes the rules of a knowledge base to be activated in forward chaining fashion, starting with the initially supplied fact base. During the inference process, the following additional assignments are made: to each derived fact the system assigns a *context* (a set of input facts necessary to infer the particular fact). These contexts are propagated through the chains of inference. The system stops if either a constraint is fired or saturation is reached. Firing a constraint means that a contradiction has occurred. The system then displays the contexts of the involved facts to the user, who is required to identify them as significant or not. The inference chain is shown for debugging or for the assertion of additional constraints.

Machine Learning Approach. Recently, machine learning was applied to knowledge base verification. For example, in Ref. 13 a verifier is developed based on the idea that examples can be generated from the given knowledge base using machine learning methods and the confirmation of these examples can be used to verify a knowledge base. Specifically, a new knowledge base is created from the original knowledge base by using machine learning techniques such that the two knowledge bases are equivalent. During this process, examples are generated from the given knowledge base. For each example, it is classified by the user and its classification is checked against the truth value of corresponding facts in the original knowledge base. If both are the same, this example is a confirmation of the given knowledge base's correctness, and the process continues. Otherwise, the original knowledge base is considered to have errors.

Logical Approach. In this approach, formal analysis is conducted based on logic theories. First a knowledge base is transformed into a set of logic formulas and then formal methods can be employed to check the correctness of the knowledge base. An example of this approach is MELODIA (20) in which the consistence of a knowledge base is established by verifying the satisfiability of a set of logic formulas derived from the given knowledge base. The satisfiability of the logic formulas can be examined by an attempt to convert the set of logic formulas into another set through some logical simplification operations. If the conversion results in an empty set then the knowledge base is considered to be consistent, otherwise, some potential inconsistency may be present in the knowledge base.

Petri-Net Approach. Due to its expressiveness and analysis power, Petri nets have been used to facilitate knowledge base verification by some researchers. The basic idea in this approach is to transform a knowledge base into a Petri-net model of some sort, and then analyze certain properties of the Petri-net model to identify potential anomalies in the given knowledge base. For example, in INDE (21), rules in a knowledge base are grouped into maximally large rule clusters or *concepts* such that the members in different concepts are mutually exclusive, which means that they cannot be fired at the same time. The concepts are transformed into Petri nets such that variables correspond to places in the graph and rules correspond to transitions. In such a Petri-net model, inconsistencies can be identified by the fact that there are places which have arcs leading to them from transitions which represent rules that assign different values to the variable associated with a place.

Meseguer's system (22) checks knowledge bases formulated in the language of propositional logic with conjunctive normal form preconditions and literals as conclusions. Monotonic forward chaining is used assuming all input facts are initially available and are consistent. Metalanguage constraints specify contradictory facts. The checking process begins by identifying subsets of rules which lead to contradictions (should unfavorable input conditions arise). Each such rule set is translated into a Petri net, where facts and rules are mapped to places and transitions respectively. Next, the system attempts to find a maximal consistent set of facts that results in a contradiction when given as input to the selected rule set. If no such consistent input assignment is found, the rule base is assumed consistent.

Knowledge-Base Reduction Approach. The idea of using knowledge base reduction to build a verifier was first proposed in Ref. 23. Theoretically, we assume two things: (1) a knowledge base is regarded as an implicit partial function whose domain is the set of all possible input values and whose range is the set of all possible conclusions and (2) each conclusion is labeled by a disjunctive normal form which represents the minimal set of all possible input values which can lead to this conclusion. Each disjunct in this form is called an environment for a conclusion which consists of symbols representing possible input values. With these assumptions, inconsistencies and redundancies in a knowledge base can be found during the process of constructing the implicit function.

Knowledge base-REDUCER2 (24) checks existing knowledge bases for consistency and redundancy. It assumes all input facts are available and rules are specified in a first-order form. Inference is assumed to progress by forward chaining under the *negation as failure* assumption. A graph is constructed with nodes representing facts and edges representing rules. Starting with the input facts ("inference level 0") the rules, whose preconditions match the input facts, form edges to facts of inference level 1, and so on until no rule is applicable. During the graph formation, every fact is assigned an "environment," that is, the set of all (minimal) input facts entailing it. A rule is redundant if its deletion causes no change in the set of derived facts. Inconsistencies are detected by facts declared as contradictory (via metalanguage constraints) but have unifiable clauses in their environments.

Metaknowledge Approach. As mentioned earlier, metaknowledge is utilized in domain-dependent verification. There are three steps in this approach: (1) determine the metaknowledge to be used; (2) represent metaknowledge in terms of integrity constraints; (3) find any violation of constraints which may occur in a knowledge base. Earlier work based on this approach can be found in Refs. 18 and 19. In a recent work (24), the knowledge about the predicates is represented in terms of semantic units which are used to describe the semantic nature of the predicates such as types and value ranges of terms, connections and relationships among predicates. Based on semantic units, a set of constraints can be derived and then violation of constraints in the knowledge base is checked.

Graph Approach. Besides Petri nets, other types of graphs such as ATMS-like (Assumption-based Truth Maintenance Systems) causal graphs, bipartite graphs, or rule dependency graphs are utilized in knowledge base verification.

KET (25) is a frame-based system. The system assigns the frame slots in a backward chaining fashion. Values of slots are either derived by rule application or are entered by the user (a slot assigned by a user input is called an "ask slot"). A graph is constructed by connecting the frame slots, with the leaves of the graph as ask slots. For each rule, there is a "path" in this graph traversing the set of slots necessary for firing that rule. The graph is checked for slots with missing or contradictory assignment rules. Rules are identified as inconsistent (redundant) if they have the same preconditions and assign different (same) values to a slot. A cycle is detected if a rule path contains the resulting slot itself. Furthermore, KET identifies overlap (redundancies) in the frame structure. If frames are found with overlapping slots, a higher-level frame is created and the overlapping slots are moved there. The overlapping frames become descendents of the newly formed frame.

Riedesel's approach (26) applies to existing rule bases specified in an attribute-value formalism with abstract set formers (quantifiers over sets). The checking algorithm works in steps, each involving a check for a certain consistency/completeness property. First, the rule base is converted into an ATMS network. Then a sequence of checking modules reasoning over the ATMS structure are called, detecting conflicts, cycles, noncovered inputs, and dead-ends. Furthermore, the structure of the knowledge base is examined to improve efficiency by clustering. These checks assume monotonicity of facts, typed variables, and provide mode specifications of variables (as input, intermediate, or output variables).

In Ref. 27 a rule dependency graph (RDG) is constructed from a given knowledge base. Improper knowledge in a knowledge base can be found by examining the RDG for certain topological structures because each type of improper knowledge forms a specific topological structure in a RDG.

Relational Approach. In this approach, we consider the union of the domains of all attributes as an attribute space and a rule as a function from the attribute space to a subset of the attribute space. Based on this idea, relations between rule functions which may reflect potential errors in a knowledge base can be defined. Therefore, the knowledge base verification can be carried out by detecting certain relations in a knowledge base. A work based on this approach can be found in Ref. 28.

Refinement Approach. As mentioned in Ref. 29 knowledge base refinement consists of three steps: (1) collecting a set of cases with known conclusions; (2) statistically analyzing the rule behavior with respect to the test cases; (3) based on the statistical behavior, refinement heuristics are applied to generate suggestions for rule refinements which can lead to a higher performance knowledge base.

In general, statistical, heuristic, or user-defined methods can be applied to a knowledge base to decrease the differences between the knowledge base evaluations and the case data provided.

MORE/MOLE (29) uses heuristics for generating refinement measurements in causal networks processed in both forward and backward chaining directions.

The user can enter three kinds of assertions: symptoms, prior-conditions, and qualifying conditions. A symptom is a condition that leads to a hypothesis if it is satisfied. A prior-condition makes a hypothesis only more or less probable and a qualifying condition influences the evidence of a symptom or a prior-condition. A knowledge base is constructed as a network with weighted vertices, with the edges constituting causal relationships. MORE processes the network in the direction of the hypotheses and compares support values with a preset threshold. Only hypotheses with values above this threshold are accepted; intermediate candidates that do not contribute to the explanation of a symptom are also rejected.

MOLE provides added refinement heuristics. It assumes that each symptom has an explanation and for each explanation as few hypotheses as possible should be used. MOLE requires the user to input *core symptoms,* for which the explaining hypotheses are found using heuristics. Additional heuristics are used to detect inconsistencies (ambiguities) and incompleteness in the network. By comparing the resultant system evaluations with test cases, missing knowledge and further ambiguities are detected: SEEK2 (29) supports the refinement of rule bases processed through backward chaining. SEEK2 understands consistency and completeness as correspondence of the system-generated results with test cases.

An initial knowledge base is run on several test cases. Based on collected performance statistics, proposals for changes in the rules are generated. Suggestions include generalizing or specializing the preconditions of a rule. SEEK2 also provides an Automatic Pilot, which applies the refinement heuristics and attempts to optimize the knowledge base using hill-climbing search. SEEK2 supports a metalanguage allowing the user to formalize refinement heuristics.

AQUINAS (31) supports the refinement of weighted decision tables (repertory grids) by comparing its results with test cases. A repertory grid is a table in which columns hold influence factors (traits) and possible solutions are entered in rows. The relationships between traits and solutions and between traits are weighted. The repertory grids of a knowledge base form hierarchical modules. AQUINAS detects analogies in the rows of the tables and helps in eliminating redundancies. The user defines constraints to provide metalanguage information. AQUINAS varies grid values and trait weights, records changes, and tries to obtain agreement with test cases by applying a hill-climbing search. Various heuristic rules are provided for selecting the grid values or trait weights to refine.

INDE (32) is applied to existing knowledge bases represented as a domain model with shallow rules specified in a Prolog-like formalism where each variable holds only one value. Rules are assumed to be processed in a forward chaining manner.

Given the domain theory and test cases, INDE constructs a Prolog-like proof for each test case. From the proof, shallow rules are generated which are usually too general. Each shallow rule is individually checked against the test data. If the result is incorrect, one or more theory rules applied in the proof are responsible. The debugger must find the theory rule(s) which have propagated the error to the shallow rules. In a metatheory, the possible errors such as *clause error* (rules are too general/restrictive) and *value error* (a numeric value is outside its valid range) are defined.

Furthermore, the metatheory computes relations between the errors (same error, opposite error). These relations form a graph; if a shallow rule generates an erroneous result, this error is propagated throughout the graph. The domain theory rules are further evaluated by error indicators measuring how frequently they are causing errors.

LAPS (33) allows the expert to input single case solutions and supports their depth-first refinement. LAPS translates resultant decision tables to data-driven rules for expert system shells like M.1 or CLIPS.

LAPS drives the development process in three sessions. First, the expert enters and solves example case data. Next, these example case data are elaborated to an initial model with heuristics, explanations, and intermediate hypotheses. Last, the examples are further developed and refined to decision tables, which are then checked for consistency and completeness using algorithmic methods.

Syntactic Inspection Approach. Knowledge base verification based on syntactic inspection approach is done by analyzing syntactic properties of a knowledge base to determine whether there are any potential errors in a knowledge base. A tool developed by this approach can be found in Ref. 34.

Incoherence Detection Approach. As mentioned in Ref. 15, facts in a knowledge base are usually associated with a set of properties such as coherence constraints, rarity of attributes, contradictory values, etc. In order to verify that a set of facts satisfies the associated properties, a predicate FC is defined on it. A set of facts is coherent if and only if FC is true on the set. A similar notion can be defined on a set of rules with regard to properties such as redundancy, conflict, and circularity. Thus, a knowledge base is statically coherent if sets of rules and facts in the knowledge base are coherent. A knowledge base is dynamically coherent if all sets of facts which can be derived from facts and rules in the knowledge base are coherent.

Traditional Software Engineering Approach. Since the software verification and validation for traditional systems have been studied by many researchers for many years, there are a lot of techniques in this area. Attempt has been made to apply them to knowledge base verification (35–40). For example, in Ref. 41 knowledge base verification is based on the checklist approach, and in Ref. 42 the assertional approach is used.

FORMAL VERIFICATION OF KNOWLEDGE-BASED SYSTEMS

Here we discuss a logic-based formalism for analyzing the properties of knowledge-based systems. In our formalism, both formal static and dynamic methods are utilized to verify knowledge-based systems for properties such as reachability, reversibility, liveness, synchronic distance, bounded fairness, and consistency.

The techniques that are normally employed to analyze the properties of knowledge-based systems include: (1) resolution refutation (for reachability, reversibility, liveness, consistency, synchronic distance, and bounded fairness); (2) model checking in temporal logic; (3) graph-theoretical algorithms for the determination of the consistency of timing constraints. Here we discuss only the first formal method.

Analysis through Resolution Refutation

To analyze a knowledge-based system's dynamic properties (i.e., functional properties that arise from the execution of the system) one can rely on either a state-space strategy or a problem-reduction strategy. The state-space strategy is based on data-driven/bottom-up forward reasoning. This reasoning employs two sets of entities. The first set is a collection of *states,* where each state reflects the condition/status of the problem at each stage on the way to its solution. The other is a collection of *operators* which help to transform the problem from one state to another. The problem-reduction strategy is based on a goal-directed/top-down backward reasoning, which again involves two sets of entities: a collection of *goals / subgoals* which describe the problem, and a collection of operators which convert a goal/subgoal into more refined, or detailed, conjunctive subgoals. It can be shown that for a problem-reduction solution of a problem, there exists an equivalent state-space solution of the same problem, and vice versa.

Our formalism adopts a problem-reduction approach when analyzing a knowledge-base that is developed using an object-oriented top-down design methodology. In modeling a system composed of a set of processes, we create a conjunctive goal (a top level activity), with each goal modeling a specific process in the system. The state of a system process is assumed to be the sum of the values of its goal arguments.

Each process is decomposed into several subprocesses, and likewise, each goal is refined into many subgoals (lower-level activities). The state of a system is thought to be the union of the states of its components processes, which corresponds to a conjunction of its goal arguments. The sole operator employed is the resolution rule.

Definition 1. A goal G is said to be reachable from a theory Θ if there is a sequence of clause selections $\sigma = c_1c_2 \ldots c_n$ that transforms G to an empty clause via the resolution rule, that is, $c_1 = G$, and $c_n = \Box$, and each c_j was obtained from earlier clauses in the sequence by the application of the resolution rule.

The set of all possible goals reachable from a theory Θ is denoted by $\Re^\star(\Theta)$. Thus, a goal G is reachable if $G \in \Re^\star(\Theta)$. We extend this concept to reachability between two goals.

Definition 2. In a theory Θ, if G is a reachable goal from Θ, a subgoal G' is said to be reachable from G if there is a sequence of clause selections $\sigma = c_1c_2 \ldots c_k$ that transforms G into $G'(c_1 = G, c_k = G')$.

All the dynamic properties of a knowledge-based system discussed in this section are verified through the construction of resolution refutations from the theory, which is derived from the knowledge-base, and a chosen goal clause.

- *Reversibility.* Determines whether an *initial goal* of a knowledge base can always be resumed. Again, we extend this concept to arbitrary goals, to confirm whether a *specific goal* can be resumed.

  A goal G is said to be *reversible* if, for each reachable goal G' in $\Re^\star(\Theta)$, G is reachable from G'.
- *Liveness.* Ensures that every goal in a knowledge-base is resumable from any other reachable goal.

  A theory is said to be *live* if, independent of the goal reached, it is possible to reach any other goal of the theory by progressing through some clause selection sequence. That is, each goal $G \in \Re^\star(\Theta)$ can be reached from any other reachable goal $G' \in \Re^\star(\Theta)$. Note that if a knowledge base is live, then all reachable goals in it are also reversible.
- *Consistency.* Analysis determines whether contradictions occur between the clauses of the underlying logic used to represent the knowledge base.

  In practice, we prefer a slightly stronger notion of inconsistency: Let G and G' be two literal goals, where $G = A(t_1, \ldots, t_n)$, $G' = \overline{A}(t'_1, \ldots, t'_n)$, for some predicate A, such that for all i, there is a substitution θ and either $t_i = t'_i\theta$ or $t_i\theta = t'_i$. A theory Θ of the underlying logic is said to be *inconsistent* if for G and G', either (1) there exist resolution refutations for both G and G'; (2) there exists a resolution refutation of G, but every sequence of clause selections $\sigma = c'_1 c'_2 \ldots c'_k \ldots$ results in an infinite recursion; (3) vice versa; or (4) every sequence of clause selections from both G and G' enters infinite recursion. A theory Θ is then said to be consistent if it is not inconsistent.
- *Synchronic Distance.* Measures the correlation between two rules, that is, their relevancy to one another. In testing and debugging a knowledge based system, it is helpful if we know the mutual dependence among the rules of the knowledge base. Synchronic distance is a metric closely related to a degree of mutual dependence between two selected clauses C_i and C_j in a theory, and is defined by

$$d_{i,j} = \max_\delta |\delta(C_i) - \delta(C_j)|$$

  where δ is a clause-selection sequence starting at any goal G in $\Re^\star(\Theta)$ and $\delta(C_i)$ is the number of times the clause C_i, i = 1, 2, is selected in δ.
- *Bounded Fairness.* This occurs if two clauses R_i and R_j of a knowledge base are in a *bounded-fair* relation, such that there is a bound on the number of times one is invoked while the other is not.

Tsai et al. (43) present a framework for software development suitable for knowledge-based systems verification and propose a formal requirements specification language that exploits the knowledge representation techniques in order to guide the specification, analysis, and development of knowledge-based systems.

SUMMARY

The knowledge-based verification tools presented earlier typically suffer from limitations with respect to the knowledge representation mechanisms and inference strategies.

1. They support only specialized and restrictive formalisms and interpreter strategies.
2. Strict assumptions are made on the available data and the inference process. For example, it is assumed that all possible facts are initially available or all rules are allowed to fire only once.

Knowledge base verification tools are usually add-ons to existing development environments for knowledge-based systems (expert system shells). Furthermore, because of the assumptions on availability of facts and domain knowledge, these techniques are applicable only after the knowledge base is nearly complete and not as tools supporting the development process. In this chapter, we have sketched a framework for the certification and evaluation of real-time safety-critical intelligent systems.

1. A high-level model allows the representation of an intelligent system independent of a particular expert system shell.
2. A nonmonotonic/temporal logic is established to serve as the semantic foundation for this model. The representational constructs (methods, demons, frames, etc.) of the language are translated into constructs of the underlying nonmonotonic/temporal logic.
3. The model, as expressed in the underlying logic, is translated into a knowledge base of a target expert system shell.
4. Criteria are developed for the certification and evaluation of intelligent systems based on this model.

This framework allows the evaluation of intelligent systems completely independent of the proposed target run-time shell. The nonmonotonic/temporal logic was chosen as the logical foundation of the framework because the widely used constructs of modern expert system shells (frames, demons, rules, objects) can be easily expressed in this logic. The certification of intelligent systems according to this framework differs from the process for conventional software systems in that the mapping from the underlying logic to the target language is more straightforward. Another difference between the certification of intelligent and conventional systems is that the semantics of our model is well-defined and better understood than that of conventional programming constructs. Therefore, more effective methods for the certification and evaluation of intelligent systems can be developed.

ACKNOWLEDGMENTS

This research is supported in part by USAF under the grant F30602-95-1-0035.

BIBLIOGRAPHY

1. C. Culbert (ed.), Special issue: Verification and validation of knowledge-based systems, *Expert Syst. Appl.,* 1990.
2. C. V. Ramamoorthy, S. Shekhar, and V. Garg, Software development support for A.I. programs, *IEEE Computer,* **20** (1): 30–42, 1987.
3. E. A. Feigengaum et al., Knowledge-based systems research and applications in Japan, *AI Mag.,* **15** (2): 29–43, Summer, 1994.
4. U. Gupta (ed.), *Validating and Verifying Knowledge-based Systems,* Washington, DC: IEEE Computer Society Press, 1991.
5. G. H. Schildt and J. Retti (eds.), *Proc. of the IFIP WG 5.4/IFAC Workshop on Dependability of Artificial Intelligence Systems,* Vienna, Austria, IFIP, 1991.
6. N. G. Leveson, Software safety: What, why, and how, *ACM Comput. Surv.,* **18** (2): 125–163, 1986.
7. C. L. Chang, J. B. Combs, and R. A. Stachowitz, A report on the expert systems validation associate (EVA), *Expert Syst. Appl.,* **1**: 217–230, 1990.
8. J.-P. Tsai and T. J. Weigert, *Knowledge-Based Software Development for Real-Time Distributed Systems,* Singapore: World Scientific, 1993, pp. 126–135.
9. P. Meseguer, Verification of multi-level rule-based expert systems. *Proc. 9th Natl. Conf. Artif. Intell.,* pp. 323–328, 1991.
10. R. Evertsz, The automated analysis of rule-based systems, based on their procedural semantics. *Proc. Int. J. Conf. Artif. Intell.,* pp. 22–27, 1991.
11. G. R. Prakash, E. Subrahmanian, and H. N. Mahabala, A methodology for systematic verification of OPS5-based AI applications, *Proc. Int. J. Conf. Artif. Intell.,* pp. 3–8, 1991.
12. D. C. Wilkins and B. G. Buchanan, On debugging rule sets when reasoning under uncertainty, *Proc. 4th Natl. Conf. Artif. Intell.,* pp. 448–454, 1986.
13. G. De Raedt, G. De Raedt, and M. Bruynooghe, Using interactive concept learning for knowledge-base validation and verification, in M. Ayel and J. P. Laurent (eds.), *Validation, Verification and Testing of Knowledge-Based Systems,* Chichester: Wiley, 1991, pp. 177–190.
14. W. R. Franklin et al., Debugging and testing expert systems. *Proc. 21st Annu. Int. Conf. Syst. Sci.,* Hawaii, pp. 159–167, 1988.
15. M. Ayel and J. P. Laurent, SACCO-SYCOJET: Two different ways of verifying knowledge-based systems, in M. Ayel and J. P. Laurent (eds.), *Validation, Verification and Testing of Knowledge-Based Systems,* Chichester: Wiley, 1991, pp. 63–76.
16. M. C. Rousset, On the consistency of knowledge bases: the COVADIS system. *Proc. 8th Eur. Conf. Artif. Intell.,* pp. 79–84, 1988.
17. R. Plant, Special issue on software quality in knowledge-based systems. *J. Syst. Softw.,* **29**: 197–198, 1995.
18. M. Suwa, A. C. Scott, and E. H. Shortliffe, An approach to verifying completeness and consistency in a rule-based expert system, *AI Mag.,* **3** (4): 16–21, 1988.
19. T. A. Nguyen et al., Knowledge base verification, *AI Mag.,* **8** (2): 69–75, 1987.
20. E. Charles and O. Dubois, MELODIA: Logical methods for checking knowledge bases, in M. Ayel and J. P. Laurent (eds.), *Validation, Verification and Testing of Knowledge-Based Systems,* Chichester: Wiley, 1991, pp. 95–105.
21. E. Pipard, Detection of incoherence and incompleteness in a knowledge base: The INDE system. *Proc. AVIGNON '88,* pp. 15–33, 1988.
22. P. Meseguer, *A New Method for Checking Rule Base for Inconsistency: A Petri Net Approach,* Tech. Rep., GRIAL 90/6, Blanes, Spain, 1990.
23. A. Ginsberg, Knowledge-base reduction: A new approach to checking knowledge base for inconsistency and redundancy. *Proc. 7th Natl. Conf. Artif. Intell.,* pp. 585–589, 1988.
24. P. Lafon, A descriptive model of predicates for verifying production systems, in M. Ayel and J. P. Laurent (eds.), *Validation, Verification and Testing of Knowledge-Based Systems,* Chichester: Wiley, 1991, pp. 149–162.
25. L. Esfahani and F. Teskey, A self modifying rule-eliciter, *Proc. Eur. Knowledge Acquisition,* pp. 16.1–16.16, 1988.
26. R. Riedesel, *Consistency and Completeness: An Exercise in Knowledge Base Validation,* Tech. Rep., Urbana-Champaign: Graduate College of the University of Illinois, 1985.
27. C. H. Wu and S. J. Lee, *Knowledge Validation for Rule-Based Expert Systems by Interpreting Dependency in Directed Graphs,* Taiwan: Department of Electrical Engineering, National Sun Yat-Sen University, 1995.
28. H. Marathe, T. K. Ma, and C. C. Liu, An algorithm for identification of relations among rules, *Proc. IEEE Int. Workshop Tools Artif. Intell.,* pp. 360–366, 1989.
29. A. Ginsberg, S. Weiss, and P. Politakis, SEEK2: A generalized approach to automatic knowledge base refinement, *Proc. IJCAI,* pp. 367–374, 1985.
30. G. Kahn, S. Nowlan, and J. McDermott, Strategies for knowledge acquisition, *IEEE Trans. Pattern Anal. Mach. Intell.,* **5** (7), 1988.
31. J. Boose, D. Shema, and J. Bradshaw, A recent progress in AQUINAS: A knowledge acquisition workbench, *Proc. Eur. Knowledge Acquisition Workshop,* pp. 2.1–2.15, 1988.
32. P. Terpstra and M. van Someren, INDE: A system for knowledge refinement and machine learning, *Proc. Eur. Knowledge Acquisition Workshop,* pp. 30.1–30.8, 1988.
33. J. Piazza, *Cases to Models Complete Expert Systems,* Tech. Rep., General Dynamics Corp., Falls Church, VA, 1990.
34. A. Preece, R. Shinghal, and A. Batarekh, Verifying expert systems: a logical framework and a practical tool, *Expert Syst. Appl.,* **5**: 421–436, 1991.
35. R. Plant and A. D. Preece, Special issue on verification and validation. *Int. J. Hum.-Comput. Stud.,* **44**: 123–125, 1996.
36. E. Plaza (ed.), Validation and verification of knowledge-based systems, *IEEE Expert,* **8**: 45–81, 1993.
37. D. Hamilton, K. Kelley, and C. Culbert, State-of-the-practice in knowledge-based system verification and validation. *Expert Syst. Appl.,* **3**: 403–410, 1991.
38. D. E. Leary (ed.), Special issue: Verification and validation of intelligent systems: Five years of AAAI workshops. *Int. J. Intell. Syst.,* **9**: 8–9, 1994.
39. A. Preece and C. Suen (eds.), Special issues: Verification and validation of knowledge-based systems, *Int. J. Expert Syst.,* **6**: 2–3, 1993.
40. N. Zlatareva and A. Preece, State of the art in automated validation of knowledge-based systems, *Expert Syst. Appl.,* **7** (2): 151–167, 1994.
41. T. Terano, A case study of expert system evaluation using a checklist-based guideline, in *AAAI-92 Workshop on Validation and Verification of Expert Systems,* Los Altos, CA: Morgan Kauffmann, 1992.
42. R. Gamble, G. C. Roman, and W. E. Ball, Formal verification of pure production system programs, *Proc. 9th Natl. Conf. Artif. Intell.,* pp. 329–334, 1991.
43. J. J. P. Tsai, T. Weigert, and H. C. Jang, A hybrid knowledge representation as a basis of requirement specification and speci-

fication analysis, *IEEE Trans. Softw. Eng.,* **SE-18**: 1076–1100, 1992.

JEFFREY J P TSAI
University of Illinois at Chicago
DU ZHANG
California State University
AVINASH SAHAY
ERIC JUAN
University of Illinois at Chicago