Avelino J. Gonzalez / Latha Ramasamy

Detecting Anomalies in Constraint-based Systems

Abstract

All software systems need to be verified, regardless of the type of techniques employed. Most existing verification tools are designed to work with conventional software or rule-based expert systems. Verification of constraint-based reasoning systems, however, has not been a popular subject of research in computer science and engineering. It could be safely said that Flannery's work on the subject [Flannery, 1993] is the first investigation done towards the objective of producing a verification tool to verify the constraints in a constraint-based system. However, her pioneering work being the first, it is rather conceptual in nature and ignores some of the more practical aspects of the problem.

The objective of this paper is to describe an automatic constraint verification tool and its testing, performed on a real world application in scheduling. The highlight of this verification tool is its ability to do open-ended inferencing to detect hidden anomalies, something Flannery's prototype was not capable of doing. In summary, this tool will ensure the consistency, completeness, and correctness of a constraint base for a constraint-based reasoning system.

1. INTRODUCTION

Acceptability of any system by its users is heavily influenced by its reliability. Verification and Validation is a formal methodology implemented to ensure this [Plant, 1996]. Such is not a one-time process but rather, a long-term one which takes place throughout the lifetime of the system. By doing this, the user's needs will be continuously met and the system will be consistent and complete all the time.

Validation determines whether the right system was built while verification determines if the system was built 'right' [O'Keefe, 1987]. It could be said that validation is 'black-box' testing and verification is 'white-box' testing [Gupta, 1993]. By black-box testing, its meant that it only determines whether the system provides a correct answer to the problem presented. Validation is not concerned about the details or correctness of the steps executed by the system to achieve this. On the other hand, white-box testing means confirming whether the system carries out the right operations to produce the desired output as specified in the detailed design. It is important to perform both verification and validation on a system because both enhance each other by detecting errors that the other is not designed to find. Since verification is the central theme of this paper, however, we only concentrate on this half of the V&V process.

1.1 Verification of Knowledge-based Systems

The verification process is composed of the following steps: 1) assurance of compliance with the requirements specification, 2) evaluation of system for consistency, 3) evaluation of system for completeness.

Compliance with the specification is largely a manual effort, and therefore, not the subject of our effort.

The knowledge in the system must be consistent with itself, and introduce no anomalies. Verification does not assume a comparison of the knowledge base with the actual domain. That is done by the validation process. It only evaluates the consistency of the system with regards to itself. Inconsistencies in a rule-base expert system manifest themselves through the following anomalies:

- **redundant rules** - two rules succeed under the same conditions and have the same conclusions.

- **conflicting rules** - two rules succeed under the same conditions but with different conclusions.
- **subsumed rules** - one rule is subsumed by another when both rules have the same conclusion but the subsumed one has additional constraints in the situation in which it will succeed.
- **unnecessary IF conditions** - two rules have unnecessary IF conditions if they have the same conclusion and a condition in one rule is in conflict with a condition in the other rule, while the remaining conditions in the two rules are otherwise equivalent..
- **circular rules** - a set of rules is circular if the chaining of these rules forms a cycle.

Affecting the completeness of a knowledge base expressed as rules are the following anomalies:

- **unreachable conclusion** - in a goal-driven system, the conclusion of a rule should either match a goal or match an IF condition of another rule in the same rule set. Failure to meet this produces an unreachable conclusion.
- **dead-end IF conditions or dead-end goals** - the attributes of the goal should either be askable (where the user provides necessary information) or match the THEN part of any of the rules. Failure to do this produces dead-end conditions or goals.
- **unreferenced attribute values** - this occurs when some value in the attribute's set of allowed values are not referenced in the IF part of any rule in the rule base.
- **illegal attribute values** - this occurs when a rule refers to an attribute value that is not in the set of legal values.

The first program created to automate the knowledge base verification is known as TEIRESIAS [Nguyen, 1987]. TEIRESIAS was designed to verify the knowledge base of MYCIN which is an infectious disease consultation system. TEIRESIAS checks the completed rule base of MYCIN and builds a rule model which reflects the connection between the attributes (which attribute(s) was used to conclude which other attribute(s)). Every time a new rule is added to MYCIN, it will be checked against the rule model for the attribute(s) found in the IF part of the rule. One of the problem with TEIRESIAS, however, is that it doesn't verify the rules used to create the initial knowledge base, i.e., the rules are assumed to be complete as they are created.

This led to development of a rule checker for ONCOCIN [Suwa, 1982]. ONCOCIN's rule checker verifies the knowledge base as it is being developed. This rule checker builds a table which displays all possible combination of attributes used in the IF conditions and the corresponding values that will be concluded in the THEN part of the rule. Later on, the table will be checked for conflicting, redundant, subsumed, and missing rules.

CHECK is another rule based knowledge base verification program [Nguyen, 1987]. It was developed to work with Lockheed Expert System (LES). CHECK does a better job at detecting rule base consistency and completeness than its predecessors. This is because it is applied to the entire set of rules for a goal and not the subset that determines the value of each attribute. Here the verification is done once the initial rule base has been created. CHECK produces dependency charts which assist in detecting the circular rule(s) and it has been designed and tested on a wide variety of knowledge bases built with a generic expert system.

1.2 Verification of Constraint-Based Reasoning.

Verification techniques for conventional software and rule-based expert systems is generally not directly applicable to constraint-based reasoning systems due to the different nature of these systems. The most trivial difference is in the architecture of the system itself [Preece, 1996], as not all of the nonconformances that exist in rule-based knowledge base exist in constraint-based knowledge base. Issues such as this contribute to the need for a separate verification system for constraint-based reasoning systems.

This led Flannery [1993] to formulate a methodology for verifying internal consistency and completeness of a constraint-based reasoning knowledge base. The nonconformance addressed in her work are 1) redundant constraints; 2) conflicting constraints; 3) subsumed constraints; 4) illegal constraints; 5) circular constraints; 6) dangling constraints; 7) over-constrained variables; and 8) unconstrained variables.

Flannery introduced these nonconformances by using the rule-based nonconformances described above as her starting point. In her work, the conflicting rules and unnecessary IF condition anomalies combine to form *conflicting constraints*. The rule-based unreachable rules and dead-end rules are likened to the constraint-based reasoning *dangling constraints*. Any requirements of the system that are not translated into a constraint is considered to be *missing constraints*.

Flannery's system was very successful in detecting nonconformances for constraints defined in a simple binary constraint satisfaction problem. However there are a few shortcomings in Flannery's system which limit its applicability. The most significant one is its inability to detect hidden anomalies, that is, anomalies not readily identifiable unless new constraints are inferred from the user-provided constraints and other inferred constraints. While Flannery's system does infer new constraints, it only does so one time. This does not enable it to detect hidden anomalies that can only be brought to light after several iterations of inferring new constraints. We refer to this capability as *open-ended constraint inference*. The tool described here enhances the Flannery system by providing the capability to do open-ended constraint inferencing

The example below would explain how the system would infer new constraints by using the Transitive Law. If, for example, there are three user-defined constraints, such as:

C1: S1 > S2
C2: S2 > S3
C3: S3 > S1

When the user-defined constraints are checked for nonconformances as pairs (i.e., C1 & C2, C2 & C3, and C1 & C3, none are detected. But C1 and C2 can infer a new constraint T1: S1 > S3, which is in conflict with constraint C3. The system described here is able infer new constraints by checking any number of user-defined constraints.

Other, less important deficiencies of Flannery's system are its inability to verify complex constraints; its limitation of only being able to accept binary constraints when the parameters are variable; and lastly, it was never tested in a real-world constraint-based system application.

2. NONCONFORMANCE DEFINITION AND THEIR DETECTION

The following sections describes every anomaly in detail as they apply to constraint-based systems. The main technique addressed by this work, that of inferring new constraints to uncover hidden and distant anomalies, is also discussed.

2.1 Nonconformance Definitions

There are several nonconformances targeted by the work described here. All of these were also addressed by Flannery rather successfully, except for the drawbacks described above. This section describes the nonconformances in greater detail.

Illegal Constraints.

A constraint is labeled as illegal if it enforces illegal conditions such as containing references to nonexistent variables; usage of invalid operators; does not follow the specified constraint or variable representation format; includes constraints with same name; includes the same variables in a constraint; includes meaningless constraints where the first parameter is not a variable; contains illegal characters in constraint definition due to typographical errors; invalid variable values; or comparison of variables with incompatible data type.

Over-constrained Variables.

A variable is said to be over-constrained if a valid value assignment is not possible for the variable because not all the constraints with that variable can be satisfied. Sometimes, all the constraints can be satisfied individually, but collectively there is no one appropriate assignment that would satisfy all the constraints. When this occurs, one or more of the constraints need to be relaxed in order to continue the process.

Unconstrained Variables.

A variable is unreferenced if no constraints exist that restrict its value. This anomaly is opposite to the over-constrained variable anomaly, and it can lead to a missing constraint anomaly. Missing constraints might cause an invalid solution to be classified as valid or a valid solution to be classified as invalid. Missing constraints can only be detected by the constraint-base developer.

Redundant Constraints.

Two constraints are labeled as redundant if they enforce the same condition in a same order or different order, as long as they are logically equivalent. There are two types of redundancies: syntactic redundancy and semantic redundancy [Gonzalez, 1993]. Syntactic redundancies occur when the constraints involved have identical conditions. Semantic redundancies occur when the logic represented by the constraints involved is the same. Syntactic redundancies are easier to detect than semantic redundancies

Conflicting Constraints.

Two constraints are said to be conflicting if they enforce opposing relationship directly or indirectly. Indirectly here means doing it through the constraint inference mechanism.

2.2 Anomaly Detection Techniques

Several techniques are used to detect anomalies in a constraint base. These are described in Flannery [1993] and will not be repeated here. However, since open-ended inferencing is the focus of this work, we will discuss this one below:

Since the relationship in a constraint could be transitive (where two constraints have one variable in common), it is possible to infer new constraints from existing constraints. This inferred constraint and a user-defined constraint can uncover a nonconformance which is not obvious otherwise. Furthermore, the inferred constraint and user-defined constraint could infer a new constraint again. This can happen as many times as there is something new to infer. Such open-ended inferrencing allows a more reliable detection for redundant constraints, conflicting constraints, and subsumed constraints. Two constraints with three variables, where one of the three being the common variable between them, could infer a third constraint which has the other two variables based on the logical operators involved.

The first constraint should be in the format C1: S1 <operator> S2 and the second constraint should be in the format C2: S2 <operator> S3. If the two constraints being evaluated are not in this form, then one of the operations below is performed prior to the inference being made.

1. reverse the order of the constraints.
2. reverse the parameters and operator in the first constraint.
3. reverse the parameters and operator in the second constraint.

An example of this includes the two constraints C1: S1 < S2, and C2: S3 > S2. Applying operation #3 above, we convert the form of C2 to C2: S2 < S3, which now fits the desired form.

Once the above operations are done if necessary then at this point the logical operators of both constraints will be evaluated based on Table 1. The symbol "?" indicates the operator indicated on the top row for S1 and S2, and the operators on the top column for S2 and S3. The operator for the inferred constraint is shown in the appropriate box in the table for the constraint

relating S1 and S3 (i.e., S1 <operator> S3). Cells which have a "-" symbol indicate that no constraint can be inferred from the existing two constraints in the above mentioned format and with a combination of operators represented by the x and y coordinate. Those cells with a logical operator in it represent the logical operator that is to be used in the resulting inferred constraint.

S1 ? S2 S2 ? S3	>	>=	<	<=	==	!=
>	>	>	-	-	>	-
>=	>	>=	-	-	>=	-
<	-	-	<	<	<	-
<=	-	-	<	<=	<=	-
==	>	>=	<	<=	==	!=
!=	-	-	-	-	!=	!=

Table 1 - Operator Inference Table

3. IMPLEMENTATION, TESTING AND EVALUATION

The evaluation of the automated constraint verification tool was done in two phases. Phase 1 tested the tool on the real-world softball league scheduling system. This evaluation included tests for redundant constraints, conflicting constraints, subsumed constraints, illegal constraints, and unreferenced constraints. Three individual tests were run to demonstrate the tool's ability to detect each type of anomaly, except for the unreferenced constraint anomaly, which only had one test case. Test cases consisted of introducing new constraints which, when combined with the existing constraints, resulted in an anomaly of some type. Furthermore, it also included the modification of some existing constraints so that a detectable anomaly resulted. The verification system successfully passed all 13 tests, thus indicating its ability to detect most important anomalies in real-world CaBR systems.

Phase 2 of the testing procedure included performing some tests on the binary constraint problem originally used by Flannery. In fact, the test cases used in Phase II represented several of the same tests run by Flannery on the same testbed. The intent of these tests was to a) verify that the verification tool could detect all the anomalies detected by Flannery's system, and b) it could detect some additional ones that were hidden and not detected by Flannery's system. Both parts of the Phase II testing were successfully met. Part b, however, was more important in that it demonstrated the power of the concept of open-ended constraint inferencing by detecting some hidden anomalies not detected by Flannery's system. Therefore, using this verification tool, distant conflicting constraints were able to be detected, which verified the advantage of open-ended inference of new constraints.

The reader is referred to [Ramasamy, 1997] for the details of the testing program, which due to limited space, cannot be adequately reflected here.

4. SUMMARY AND CONCLUSIONS

It is very important to be able to infer new constraints from the user-defined constraints whenever possible. Inferring new constraints is a crucial feature in a verification tool because it allows the detection of hidden anomalies. This verification tool is productive because it can infer new constraints from user-defined constraints. The hidden anomalies are those ones that cannot be detected by superficially analyzing the constraints because a user-defined constraint can be redundant, subsumed, or conflicting when it is evaluated with a inferred constraint. This verification tool does open-ended inferencing, so it is able to infer as many new constraints as allowed by the resources.

This tool successfully implements open-ended inferencing of new constraints to detect hidden anomalies that would otherwise go undetected.

5. REFERENCE

[Flannery, 1993] Flannery, L. M., "Detecting Nonconformances in a Constraint-Based Reasoning System Knowledge Base", Master's thesis, Department of Computer and Electrical Engineering, University of Central Florida, July 1993.

[Gonzalez, 1993] Gonzalez, A. J., and Dankel, D. D., The Engineering of Knowledge-based Systems - Theory and Practice, Englewood Cliffs, NJ: Prentice Hall, 1993.

[Gupta, 1993] Gupta, U. G., Validation and Verification of Expert Systems, Los Alamitos, CA: IEEE Press, 1993.

[O'Keefe, 1987] O'Keefe, R. M., Balci, O., Smith, E. P., "Validating Expert System Performance", IEEE Expert, Vol. 2, No. 4, 1987.

[Nguyen, 1987] Nguyen, T. A., Perkins, W. A., Laffey, T. J., Pecora, D.,"Knowledge base Verification", AI Magazine, Vol. 8, No. 2, Summer 1987.

[Plant, 1996] Plant, R., Preece, A. D., "Editorial: Special Issue in Verification and Validation", International Journal of Human-Computer Studies, pp. 123-125, 1996.

[Preece, 1996] Preece, A. D., "Validating dynamic properties of rule-based systems", International Journal of Human and Computer Studies, pp. 145-169, 1996.

[Ramasamy, 1997] Ramasamy, L., "Open-ended Inference of New Constraints in Constraint Verification", Master's Thesis, Department of Electrical and Computer Engineering, University of Central Florida, 1997 (To be published).

[Suwa, 1982] Suwa, M., Scott, C. A., Shortliffe, E. H., "An Approach to Verifying Completeness and Consistency in a Rule-Based Expert System", AI Magazine, Vol. 3, No. 4, Fall 1982.

Avelino J. Gonzalez, Professor
ECE Department, University of Central Florida
PO Box 162450; EN 407
Orlando, FL 32816-2450
ajg@ece.engr.ucf.edu
FAX: + 407-823-5027

Latha Ramasamy
ECE Department, University of Central Florida
PO Box 162450; EN 407
Orlando, FL 32816-2450