Improving Knowledge-Based System Performance by Reordering Rule Sequences

Neli P. Zlatareva

Department of Computer Science
Central Connecticut State University
1615 Stanley Street
New Britain, CT 06050
E-mail: zlatareva@ccsu.edu

Abstract

In this paper, we argue that KBS validation should not be limited to testing functional properties of the system, such as its input - output behavior, but must also address its dynamic properties, such as its run-time performance. We describe an automated procedure, which under certain limitations can recognize relations between rules, typically expressed as "meta-rules" or "control heuristics" and hard-wired in the KBS's control strategy. The presented procedure takes as an input the operational version of the knowledge base theory generated by a CTMS-based verification tool, and returns a reordered set of rules which when applied to a specific problem generates possible solutions of that problem in order according to a specified criterion, for example the length of the path leading to the solution. We show an example to illustrate the proposed procedure.

Introduction

Knowledge-Based System (KBS) validation aims to evaluate and improve system performance. The two most important performance measures are predictive accuracy and run-time efficiency. Predictive accuracy is typically defined by running test cases with known solutions once the system is proved to be structurally correct. A KBS is considered valid if it correctly solves all test cases, and there is evidence that it will correctly solve any other case presented to the system. Being valid is a required property of a KBS; however, it is rarely a sufficient one (except in small toy domains). In most cases, KBSs encounter intractable computational tasks where a run-time efficiency is essential for system acceptability. In such applications, another dimension must be added to the V&V process, namely assuring the maximum efficiency of the computation process. This aspect of KBS validation has not been sufficiently studied yet, in part because it was believed that a run-time efficiency is a property of the control strategy that utilizes domain-dependent knowledge to guide the reasoning process [Fenzel & Straatman, 1998]. Examples of such knowledge include the order in which observations are obtained during the diagnostic process, the order in which components are configured during the design process, etc. Acquiring control knowledge from domain experts, however, is the most difficult component of the knowledge acquisition process. It is unrealistic to expect that experts can generalize case-dependent heuristics or define the certainty value of a single rule in isolation. This is why it is essential to equip the validation tool with an automated procedure, which can recognize problem-dependent relations between rules and reorder rule sequences according to these relations so that to guarantee the maximum efficiency of the computation process.

In this paper, we present a method for rule reordering, which can be easily performed as part of the validation process. Similar idea is discussed in [Terano & Kobayashi, 1995], where a method based on a genetic algorithm is proposed to improve system performance by changing certainty values associated with the rules. These changes require test cases, which are used as "oracles" in the refinement process. Our method does not require test cases, although it assumes that the system has undergone structural and functional testing, and all detected anomalies have been removed from the knowledge base.

We first discuss the motivation for this research and related work, and then outline the basic algorithm for computing and reordering rule sequences. An example is included to illustrate the proposed method. We conclude with some preliminary results and discuss our plans for future work.

Problem definition, motivation and practical justification

As stated in [Groot, ten Teije & van Harmelen, 1999], there are two separate although related tasks in KBS validation:

1. Validation of functional properties, which is concerned with the input – output behavior of the system.
2. Validation of dynamic properties, which is concerned with the computation process itself.

Functional validation deals with "how" the solution is computed, while dynamic validation deals with "what" counts as a solution. The authors suggest that dynamic validation is a refinement of functional validation, because two systems with the same dynamic properties have necessarily the same input – output behavior, whereas two implementations with the same input – output behavior may have different computational models.

To illustrate the difference between functional and dynamic properties of a KBS, consider a medical application where the goal of the KBS is to find the best way to prove a certain diagnosis relative to a specific environment. Because the environment can be different, the system is expected to contain alternative computational models and apply them selectively. The extended example presented below can be interpreted as a representation of a relation between an observation, S, intermediate tests A through I, some of which are mutually exclusive, and the final diagnosis, F. There are four possible ways to compute F. The shortest solution, which in many cases may be the preferred one, utilizes R8, R11, and R12. In some cases, however, some tests (I, for example) may not be available or possible, which is why the KBS must identify R1 → R7 → R9 → R10 as an alternative path to the goal. Functional validation is intended to show that the system satisfies the input – output specification (i.e. given S, F can be proved), while dynamic validation evaluates the computation process itself and recognizes R8 → R11 → R12 as the shortest solution of the stated problem.

Recognizing the two sides of the validation process is essential for developing tools intended to improve the dynamic properties of a KBS. Run-time efficiency is the most important dynamic property, which depends on the order in which rules fire in the problem-solving process. This order, in turn, is dependent on many factors: the chaining method, the conflict resolution strategy, whether or not a non-monotonicity is incorporated into the knowledge model, etc. (see [Boswell & Craw, 1999] for discussion and examples on how these factors influence the problem-solving process and can be accounted for in knowledge refinement). Under certain limitations, however, forward and backward chaining methods produce exactly the same results. In such cases, the only difference in the

generated outcome comes from the way the goal of a KBS is defined, i.e. whether the system is expected to generate "the shortest", "the first", or "all" solutions to the presented problem. In many applications, the user is interested not only in "the shortest" solution, but also in the time in which this solution is reached and the resources needed for its implementation. In both cases, the system is expected to find the most efficient solution (i.e. the solution with the minimum number of the rules fired in the computation process), and upon request offer alternative solutions in decreasing order of their desirability. We discuss next how the system's dynamic behavior can be evaluated and refined as part of the validation process.

Computation and reordering of rule sequences

One of the advantages of a production rules model is that rules can be entered in the knowledge base as they arrive, because their order is irrelevant to the predictive accuracy of the KBS. This may considerably ease the knowledge acquisition process, but even for small-scale applications may lead to an intractable problem-solving process. If there exist problem-dependent relations between rules (control knowledge), instead of relying on domain experts to formulate and formalize such relations in a form of control rules (thus complicating not only the knowledge acquisition process, but also the control strategy), the validation tool may automatically recognize such relations and suggest changes in rule order to improve the run-time performance of the KBS. Because this process is viewed as part of the validation framework, the system must have undergone structural and functional testing beforehand. For that purpose, we use a CTMS-based tool, the VVR system [Zlatareva, 1994]. One advantage of the CTMS-based tool is that all potential solutions are explicated during structural verification, and can be used as an input for the proposed dynamic validation procedure. The desired output of that procedure is a reordered set of rules, which generates the solutions of a presented problem in the order of their desirability (according to a specified criterion, for example the length of a rule sequence). This is exactly the opposite of what the so-called "anytime algorithms" [Russell & Zilberstain, 1991] aim to achieve. Any-time algorithms gradually approach the perfect solution; the more time the algorithm has, the more are its chances to find it. Our intension is to make it possible for a KBS to identify the best with respect to the existing context solution early in the computation process by recognizing and implementing problem-dependent relations between rules, which guide the computation process towards it.

The proposed dynamic validation procedure utilizes the following algorithm. The input is an operational theory produced by a CTMS-based verification tool, the initial rule set and a specific problem for which the input – output specification is provided, and the output is a reordered rule set which embodies an implicit control forcing the KBS's inference engine to generate problem solutions in order according to their length. We assume that KBS inference method is forward or backward chaining, and no explicit control strategy forces rules to fire "out-of-order" (i.e. rules that come first, fire first).

1. Using the operational theory produced by a CTMS-based verification tool, identify all potential solutions of a given problem. A solution is a rule chain leading from initial facts to a specified final hypothesis.
2. Sort rules into levels corresponding to the steps of the operationalization process at which a given rule was applied (see the example below).
3. Identify alternative paths to intermediate and final hypotheses and compute their length counting the number of rules in each path.

4. For each alternative path starting with the shortest one, move the end-rule after all lower-level rules.

We illustrate this algorithm on an example presented next.

Example

Consider the following set of rules. Let S be the input, and F be the final hypothesis.

R1: S → A
R2: S → B
R3: B → H
R4: H → E
R5: A → E
R6: E → I
R7: A → D
R8: S → C
R9: D → J
R10: J → F
R11: C → I
R12: I → F

Forward chaining and backward chaining, both, generate the same solution: R1, R7, R9, R10, and R10, R9, R7, R1, respectively. A shorter path to the final hypothesis, F, however, is R8, R11, R12. To get this shorter solution, rules must be reordered as described in the previous section.

1. Identification of alternative solutions. Intermediate results of the operationalization process produced by a CTMS-based verification tool are given below. It is easy to see that there are alternative paths to intermediate hypotheses E and I, and to the final hypothesis, F, respectively.

Step 1 A: (S, R1)
B: (S, R2)
C: (S, R8)

Step 2 H: (B, R3)
E: (A, R5)
D: (A, R7)
I: (C, R11)

Step 3 E: (H, R4)
I: (E, R6)
J: (D, R9)
F: (I, R12)

Step 4 F: (J, R10)

2. Sorting rules into levels according to the step at which each rule was applied in the operationalization process.

Level 1: R1, R2, R8
Level 2: R3, R5, R7, R11
Level 3: R4, R6, R9, R12
Level 4: R10

3 & 4. Identification of alternative paths. For each alternative path starting with the shortest one, move the end-rule after all lower level rules.

Alternative paths to E:

- P1: S → **R1** → A → **R5** → E
- P2: S → **R2** → B → **R3** → H → **R4** →E

To find the shortest path to E, the inference engine must fire R5 before R3. R5 however must follow R2, because R2 is a lower-level rule.

Alternative paths to I:

- P1: S → **R1** → A → **R5** → E → **R6** → I
- P2: S → **R2** → B → **R3** → H → **R4** → E → **R6** → I
- P3: S → **R8** → C → **R11** → I

Because the length of the path, P3, is less then the length of P1, R11 must be moved before R5 to ensure that P3 is generated first.

Alternative paths to F:

- P1: S → **R1** → A → **R5** → E → **R6** → I → **R12** → F

- P2: S → **R2** → B → **R3** → H → **R4** → E → **R6** → I → **R12** → F
- P3: S → **R8** → C → **R11** → I → **R12** → F
- P4: S → **R1** → A → **R7** → D → **R9** → J → **R10** → F

Here the length of P3 is less than the length of P4. Therefore, R12 must fire before R9, but after R7, because R7 is a lower-lever rule.

After introducing all changes, the reordered rule sequence becomes: R1, R2, R8, R11, R5, R3, R7, R12, R4, R6, R9, R10. Now the shortest solution, R8 → R11 → R12, will be generated first in just one pass through the rule set. If alternative solutions are requested, R1 → R7 → R9 → R10 will be generated next also in just one pass. The other two solutions, R1 → R5 → R6 → R12 and R2 → R3 → R4 → R6 → R12, require two passes through the rule set and will be generated last.

Conclusion

We have argued in this paper that KBS validation should not be limited to assuring functional properties of the system, such as its input - output behavior, but must also address its dynamic properties, such as its run-time performance. We have shown that under certain limitations, an automated procedure can recognize relations between rules, which are typically expressed as "meta-rules" or "control heuristics" and hard-wired in the KBS's control strategy. One advantage of having such a procedure in place is that the knowledge acquisition process will be simplified, because acquiring control knowledge from domain experts is the most subjective and difficult task.

At present, the proposed procedure was applied to small examples, but we believe that it will be of practical interest for real-world applications. More work remains to be done to see to what extend dynamic validation can substitute for traditional methods for acquiring control knowledge.

References

Craw, S.; Boswell, R. 1999. Representing Problem Solving for Knowledge Refinement. In *Proceedings of the 16-th National Conference on Artificial Intelligence (AAAI'99),* AAAI Press.

Fenzel, D.; Straatman, R. 1998. The Essence of Problem-Solving Methods: Making Assumptions for Gaining Efficiency. *International Journal of Human-Computer Studies*, 48(2), pages 181 – 215.

Groot P.; ten Teije A.; van Harmelen, A. 1999. Formally Verifying Dynamic Properties of Knowledge Based Systems. In *Proceedings of the 11-th European Workshop on Knowledge Acquisition, Modeling and Management (EKAW'99).* Lecture Notes in AI, Springer Verlag.

Russell, S.; Zilberstain, S. 1991. Composing Real-Time Systems. In Proceedings of the 12-th International Joint Conferences on Artificial Intelligence (IJCAI'91*),* Boston, pages 212 – 217.

Terano, T; Kobayashi. K. 1995. Changing the Traces: Refining a Rule Base by Genetic Algorithm. In Proceedings of the IJCAI'95 Workshop on Validation and Verification of Knowledge-Based Systems. Montreal, Canada.

Zlatareva, N. 1994. A Framework for Verification, Validation and Refinement of Knowledge Bases: the VVR System. *International Journal of Intelligent Systems*, 9(8), pages 703 –738.