Explicating Semantic Relations in Non-Monotonic Theories to Facilitate Validation Analysis

Neli P. Zlatareva

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

Abstract

The inherent vagueness and ambiguity of non-monotonic reasoning makes it impossible to formulate detailed specifications to validate KBS performance by using traditional test-case-based approach. In this paper, we present a practical validation technique for non-monotonic KBSs, which utilizes automatically generated test cases instead. We show how such test cases and other useful validation knowledge can be acquired from operational theories, which are functionally equivalent to the original non-monotonic theory, and how this validation knowledge can be used by domain experts to evaluate the validity of the knowledge model.

Introduction

Validation is a critical step in Knowledge-Based Systems (KBSs) development, which aims to demonstrate that the system behaves as intended. There are two well-defined approaches for validating software systems in general, known as the "black-box" and the "white-box" testing. Black-box testing ignores the internal workings of the system and specifies performance requirements as test cases with known solutions. White-box testing, on the other hand, exercises the internal structure of the system and explicates the reasoning path leading to the generated output. In the context of KBSs, white-box testing is associated with knowledge base *verification* which aims to prove that the system's knowledge is structurally correct, while black-box testing is associated with system *validation* which aims to prove that the KBS as a whole performs correctly.

While verification can be easily automated (Zlatareva and Preece, 1994), validation places a heavy burden on domain experts, who are ultimate decision-makers about whether the KBS meets performance requirements. In this paper, we propose a white-box validation technique, which aims to reduce the validation load on domain experts by providing them with automatically generated input - output specifications and other semantic relations underlying the knowledge model. That is, instead of being caught up with crafting a carefully designed set of test cases, domain experts are presented with automatically generated specifications, which they evaluate to determine the functional correctness of the KBS.

The idea of using automatically generated test cases for KBS validation is not new. It has been successfully implemented and applied for validation of traditional rule-based systems (Vignollet and Lelouche 1993; Zlatareva, 1993). Here we show how this idea can be applied to KBSs incorporating vague, incomplete or commonsense knowledge. Due to the complexity of such systems, producing detailed performance specifications, required for typical black-box validation, is a very difficult or even impossible task. As suggested in (Jarrold, 2003), "validation of intelligence for commonsense systems should be more like giving a psychological test than doing a proof of correctness".

The paper is organized as follows. First, we briefly elaborate on the existing approaches to KBS validation, stressing the need for less stringent validation technique suitable for non-monotonic and other types of systems with vague and ambiguous knowledge. Next, we review the logical framework introduced in (Zlatareva, 2004), which is used as a test-bed for the proposed validation analysis. We use an extended example to show how to systematically explicate a complete set of input -- output specifications and other semantic relations underlying the knowledge model, and how these specifications can be used by domain experts to evaluate KBS performance.

Why test-case-based validation does not work for non-monotonic KBSs

Traditional KBS validation employs the black-box testing, and thus depends on the availability of a carefully selected set of test cases with known solutions. By running these test cases and comparing the results generated by the system to the expected ones, domain experts judge about the quality of KBS performance. Unfortunately, there are few situations where the results generated by the system can be determined as objectively correct, especially if the system exploits assumptions and/or ambiguous knowledge in its reasoning. The success of the black-box testing in such cases strongly depends on the quality of the test cases themselves. As pointed out in (Tsuruta, 2002), validation knowledge is different than domain knowledge, and acquiring it must be carried out independently from the mainstream KBS development. This puts an extra burden on domain experts, who are commonly recognized as a “bottleneck” in KBS development. On the other hand, validation is an on-going process. Even if validation knowledge is precisely acquired and processed, it reflects the desired final state of the knowledge model, and may not be helpful at earlier stages when detecting and correcting performance errors may be easier.

In this paper, we describe a white-box validation technique, which does not require explicit validation knowledge. Instead, it derives validation knowledge in a form of *synthetic test cases* by simulating the behavior of the KBS and explicating hidden semantic relations in the knowledge model. These explicated relations are then evaluated by domain experts to determine their validity. It is important to note that the targeted relations explicated during the validation process are different than ones of interest for knowledge base verification. Verification is concerned with structural correctness of the model, which is why it analyzes every step of explicated reasoning paths. Validation, on the other hand, assumes that the knowledge model is syntactically correct and concentrates on the semantic aspect of it, namely on input – output specifications and other hidden semantic relations implied by underlying assumptions.

We argue in this paper that white-box validation has two major advantages:

(i) it decreases the validation load on domain experts, and

(ii) evaluating hidden semantic relations at early stages of KBS development makes it easier to identify incorrect chunks of the knowledge model, and eventually helps domain experts discover unexpected consequences of their own knowledge.

Further in this paper, we show how the proposed validation technique can be carried out within a logical framework intended for structural verification (Zlatareva, 2004). The backbone of the latter is a conversion algorithm, which translates the original non-monotonic theory into a set of monotonic sub-theories whose stable extensions (transitive closures) are functionally equivalent to possible extensions of the original theory. What we gain from this conversion is that now we deal with monotonic theories, which are easily amenable to formal analysis. As we show next, the operational versions of these theories contain all data dependencies for acquiring synthetic test cases and other useful semantic relations to help domain experts evaluate the validity of the underlying knowledge model.

The representation framework and the conversion algorithm: an overview

We assume that the non-monotonic theory under validation consists of the following very general default rules:

Ri: (premise_1, … , premise_n)
(assumption_1, … ,assumption_m)$\rightarrow$
conclusion_i

where:

- (premise_1, …, premise_n) is the set of monotonic supporters for conclusion_i, which we call for short the *In-list*, (terminology adapted from (Doyle, 1979)).

- (assumption_1, …, assumption_m) is the set of non-monotonic supporters for conclusion_i, called the *Out-list*.

- conclusion_i holds if and only if all of its monotonic supporters hold, and none of its non-monotonic supporters holds. We assume that conclusions may serve as both monotonic and non-monotonic supporters for other conclusions.

As described in (Zlatareva 2004), non-monotonic theories of this type can be converted to a set of functionally equivalent monotonic theories, which are more easily amenable to formal analysis. The first step in the conversion process is to map each non-monotonic rule into one or more monotonic rules as follows:

1. If the original rule has an empty assumption list and its conclusion does not belong to the *Out-list* of any other conclusion, then the monotonic version of it is the same as the original rule.

2. If the original rule has an empty assumption list, but its conclusion serves as a non-monotonic supporter for another conclusion, then it is converted into a set of monotonic rules some of which capture the implicit semantic constraints between the negation of the rule's premises and the negation of the rule's conclusion. Such constraints must have a real domain counterpart, although these types of "negative" relations are typically not an explicit part of the knowledge acquisition process. They, however, may provide useful validation knowledge, as we show further in this paper.

3. If the original rule has a non-empty *Out-list*, then it is translated into a set of monotonic rules, which reflect both the negated relation between the assumptions and the rule's conclusion, and the negated relation between the latter and the conclusions of the rules, where original rule conclusion serves as a premise.

To illustrate the conversion process, consider the following non-monotonic theory and its monotonic equivalent:

R1: (A1, A2) ( ) → C1
R2: (C1, A3) ( ) → C2
R3: (A4, A5) (A8, A9) → C2
R4: (C2, A6) (C1) → C3
R5: (A1, A3) (C1) → C8
R6: (C4, C2) ( ) → C5
R7: (C5, A8) ( ) → C6
R8: (C6, A9) ( ) → C7
R9: (A7) (C3) → C4
R10: (A4, C8) (A8, C4) → C9

The resulting monotonic theory is the following:

R1*: A1 $\wedge$ A2 → C1
R1A*: $\neg$A1 → $\neg$C1
R1B*: $\neg$A2 → $\neg$C1
R2*: C1 $\wedge$ A3 → C2
R3*: A4 $\wedge$ A5 $\wedge$ $\neg$A8 $\wedge$ $\neg$A9 → C2
R3A*: A8 → $\neg$C2
R3B*: A9 → $\neg$C2
R4*: C2 $\wedge$ A6 $\wedge$ $\neg$C1→ C3
R4A*: C1 → $\neg$C3
R4B*: $\neg$C2 → $\neg$C3
R5*: A1 $\wedge$ A3 $\wedge$ $\neg$C1 → C8
R5A*: C1 → $\neg$C8
R6*: C4 $\wedge$ C2 → C5
R7*: C5 $\wedge$ A8 → C6
R8*: C6 $\wedge$ A9 → C7
R9*: A7 $\wedge$ $\neg$C3 → C4
R9A*: C3 → $\neg$C4
R10*: A4 $\wedge$ C8 $\wedge$ $\neg$A8 $\wedge$ $\neg$C4 → C9
R10A*: A8 → $\neg$C9
R10B*: C4 → $\neg$C9

By identifying consistent subsets of assumptions defining possible extensions of the original non-monotonic theory, the resulting monotonic theory can be divided into a set of sub-theories, which are functionally equivalent to the possible extensions of the non-monotonic theory. These consistent subsets are easy to identify if we consider logical constraints on the inputs of the monotonic theory. There are four of them here (namely, {A1, ¬A1}, {A2, ¬A2}, {A8, ¬A8}, {A9, ¬A9}), dividing the input data set into 16 logically consistent subsets of assumptions (inputs). Therefore, there are 16 monotonic sub-theories that must be independently verified as described in (Zlatareva 2004), before the validation step is initiated. Assuming that all verification tests were successful, operational versions of monotonic sub-theories (called *grounded stable extensions,* or GSEs) describe each conclusion (output) in terms of assumptions (inputs) and rules contributing to its derivation (Zlatareva and Preece 1994). Slightly transformed to capture the semantics of the original non-monotonic theory, these input – output mappings are in fact synthetic test cases, which domain experts can utilize to evaluate the validity of the knowledge model. Next, we discuss how synthetic test cases and other hidden semantic relations embedded in GSEs are derived and interpreted to facilitate validation analysis.

Acquiring synthetic test cases and semantic constraints from GSEs

Typical black-box testing exercises test cases mapping directly input data sets to outputs. We refer to such mappings as *high-level test cases*. Mappings from input data sets to intermediate conclusions may also provide helpful validation information, especially if evaluation of high-level test cases revealed errors in the knowledge model. We refer to such mappings as *low-level test cases.*

Acquiring high-level test cases from GSEs is easy, because all potentially reachable conclusions are represented in terms of input data sets originating them. For example, GSE originated by the inputs {A1, A2, A3, A4, A5, A6, A7, A8, A9} contains the following dependencies:

C1: (A1, A2),
C2: (A1, A2, A3),
¬C2: (A8),
¬C2: (A9),
¬C8: (A1, A2),
¬C9: (A8),
¬C3: (A1, A2),
¬C3: (A8),
¬C3: (A9),
C4: (A7),
¬C9: (A7),
C5: (A7, A1, A2, A3),
C6: (A7, A1, A2, A3, A8),
C7: (A7, A1, A2, A3, A8, A9).

Assuming that the theory has been successfully verified, the following semantic inconsistencies derivable from here hold (otherwise, the theory would contain logical contradiction $C2 \wedge \neg C2$):

$\neg (A1 \wedge A2 \wedge A3 \wedge A8)$,
$\neg (A1 \wedge A2 \wedge A3 \wedge A9)$.

Let C7 be a final conclusion in our example theory. The following high-level test case for C7 is derived from the GSE referenced above:

(A1, A2, A3, A7, A8, A9) ( ) → C7

If domain experts agreed that {A1, A2, A3, A8} and {A1, A2, A3, A9} are invalid inputs (which is why $C2 \wedge \neg C2$ is a pseudo-contradiction, but not a real one), the detected test case for C7 may suggest a performance error in the original theory. Whether this is the case, is up to domain experts to decide, but an interesting relationship can indeed be noticed between R3, R7, and R8. The former suggests, that C2 is *Out*, if A8 or A9 is *Out*. But, both assumptions are required to be *In* for C7 to be derived, which also implicitly depends on C2.

Additional, more focused, information about potential errors in the knowledge model can be obtained from reviewing low-level test cases mapping inputs to intermediate conclusions. Here are relevant low-level test cases (positive and negative), acquired from the GSE referenced above:

A1, A2 → C1
A1, A2, A3 → C2
A7 → C4
A7, A1, A2, A3 → C5
A7, A1, A2, A3, A8 → C6
A1, A2 → ¬C8
A8 → ¬C2
A9 → ¬C2
A8 → ¬C9
A7 → ¬C9

Negative test cases state counter-arguments for unreached conclusions. Consider the last two relations implying ¬C9. Although the first one seems trivial, because A8 is a direct non-monotonic supporter for C9, the second one defines an interesting counter-argument for C9, which is not explicitly stated in the original theory. Acquiring negative test cases is especially difficult task for domain experts. Reviewing automatically generated negative test cases, especially for theories employing assumptions, may reveal potentially unknown or unexpected dependencies prompting domain experts to revise their own knowledge.

Similarly, we acquire high-level and low-level test cases and semantic constraints from all grounded stable extensions, generated during the conversion process. Many of these dependencies will be duplicated in different extensions. After filtering all duplications, below are shown relevant test cases capturing the semantics of our example non-monotonic theory.

Positive high-level test cases:

(A1, A2, A3, A7, A8, A9) ( ) → C7
(A1, A3, A4, A5, A6) (A2, A8, A9) → C9

Positive low-level test cases:

(A1, A2) ( ) → C1
(A1, A2, A3) ( ) → C2
(A1, A3) ( ) → C2
(A4, A4) (A8, A9) → C2
(A4, A5) (A1, A8, A9) → C3
(A4, A5) (A2, A8, A9) → C3
(A4, A5, A6) (A2, A8, A9) → C3
(A4, A5, A6) (A1, A8, A9) → C3
(A6) (A1) → C3
(A6) (A2) → C3
(A7) ( ) → C4
(A7, A8) ( ) → C4
(A7, A9) ( ) → C4
(A1, A2, A7) ( ) → C4
(A1, A2, A3, A7) ( ) → C5
(A1, A2, A3, A7, A8) ( ) → C5
(A1, A2, A3, A7, A9) ( ) → C5
(A1, A2, A4, A5, A7) (A8, A9) → C5
(A1, A2, A3, A7, A8) ( ) → C6
(A1, A3) (A2) → C8

Automatically generated set of test cases can be further filtered by eliminating those, whose sets of assumptions are supersets of another test case's assumptions, given that both test cases have the same set of premises.

Negative high-level test cases can be helpful as well. Here are negative high-level test cases for C9 explicating the inputs under which C9 is not reachable:

(A9) ( ) → ¬C9
(A1, A2, A7) ( ) → ¬C9
(A7, A8) ( ) → ¬C9
(A7, A9) ( ) → ¬C9

Relations shown next, explicated by other GSEs of our example theory, can be viewed as negative low-level test cases, and may occasionally provide domain experts with additional validation information.

¬A1 → ¬C1,
¬A2 → ¬C1
A1, A2 → ¬C3,
A8 → ¬C3,
A9 → ¬C3
A4, A5, A6, ¬A2, ¬A8, ¬A9 → ¬C4,
A4, A5, A6, ¬A1, ¬A8, ¬A9 → ¬C4,
A4, A5, ¬A1, ¬A8, ¬A9 → ¬C4,
A4, A5, ¬A2, ¬A8, ¬A9 → ¬C4,
A6, ¬A1 → ¬C4,
A6, ¬A2 → ¬C4
A1, A2 → ¬C8

Synthetic test cases and semantic constraints, acquired from GSEs, provide domain experts with valuable validation information. In addition, they may reveal relations and unnoticed or unexpected results provoking experts to expand or revise their own domain knowledge. In our example theory, it is easy to see that the input set {A1, A2, A3, A4, A5, A6, A7, A8} does not imply a final conclusion. The acquired semantic constraints explain why none of the final hypotheses is derived. This analysis suggests that either this data set represents an invalid input (which is up to domain experts to decide), or the theory is incomplete and needs to be revised.

Conclusion

In this paper, we have presented a white-box validation technique intended to help domain experts validate non-monotonic KBSs. We argued that:

(i) validation of such systems is by default less rigorous than validation of monotonic systems,

(ii) the traditional test-case-based approach is not practical, because of the inherent vagueness and ambiguity of non-monotonic reasoning, which makes it impossible to formulate detailed performance specifications, and

(iii) utilizing automatically generated synthetic test cases not only decreases the validation load on domain experts, but also can help them to discover unexpected consequences of their own knowledge.

The proposed validation analysis utilizes the logical framework initially proposed for structural verification (Zlatareva, 2004), which makes it possible to build a uniform testing tool for non-monotonic KBSs evaluation.

Although we have tested the proposed validation technique on several example knowledge bases (50 – 70 rules), and the experiments revealed interesting relations among non-monotonic rules, more work remains to be done to determine the complexity of this analysis when applied to large-scale KBSs.

References

Doyle, J. 1979. Truth Maintenance Systems. *Artificial Intelligence* 12: 231 – 272.

Jarrold, W. 2003. Using an Ontology to Evaluate a Large Rule Based Ontology: Theory and Practice. In *Performance Metrics for Intelligent Systems PerMIS '03* (NIST Special Publication 1014).

Tsuruta, S., T. Onoyama and Y.Taniguchi 2002. Knowledge-based Validation Method for Validating Intelligent Systems. In *Proceedings of FLAIRS'2002*, AAAI Press.

Vignollet, L. and R. Lelouche 1993. Test Case Generation Using KBS Strategy. In *Proceedings of IJCAI'93*, Chambery, France.

Zlatareva, N. 1993. Distributed Verification and Automated Generation of Test Cases. In *Proceedings IJCAI'93 Workshop on Validation of KBS*, Chambery, France.

Zlatareva, N., and Preece A. 1994. An Effective Logical Framework for Knowledge-Based Systems Verification. *International Journal of Expert Systems: Research and Applications*, vol. 7, No 3.

Zlatareva, N. 2004. Testing the integrity of Non-Monotonic Knowledge Bases Containing Semi-normal Defaults. In *Proceedings of FLAIRS'2004,* AAAI Press.