Experiences of using Verification Tools for Maintenance of Rule-Based Systems.

Mark A. Dahl
Keith Williamson

The Boeing Company
P.O. Box 24346, MS 73-43
Seattle, WA 98124-0346

Abstract

This paper reports on the use of tools that statically analyze rule-bases in order to detect logical errors, such as redundancy, inconsistency, and incompleteness. The analysis uses a variant of the KB-Reducer algorithm (Ginsberg 1991). In contrast to accounts of similar tools, which seldom describe tools as used in the field, the experience described in this paper is derived from using tools to analyze a large suite of knowledge bases in production use at Boeing. The tools serve to augment the ongoing process of knowledge base maintenance. Common causes of errors detected are reported. This paper shows that making such tools practical is non-trivial but rewarding.

Introduction

As observed in Preece and Shinghal, 1992, tools for finding anomalies in knowledge bases (KBs) have shown sufficient potential in KB verification and validation, to motivate researchers to build a series of tools over the past decade. Although Preece's paper provides some data on the practical aspects of using such tools on KBs intended for production use, it also notes the general lack of such data. The goal of this paper is to provide additional experiential data for the benefit of potential tool builders and users.

This paper describes experiences encountered using an anomaly detecting tool called KBR3, in maintaining a suite of knowledge bases that comprise a production application called Engineering Standards Distribution System (ESDS). ESDS currently has 50 KBs in production which accumulatively contain tens of thousands of rules. Results of 7 KBs are covered. An overview of the ESDS project can be found in Dahl, 1993. An overview of the motivation for adopting a maintenance strategy which includes the KBR3 tool is described in Dahl and Williamson, 1992.

The rest of this paper is organized as follows. An overview of KBR3's basic functionality is given, followed by a description of error types detected by KBR3. Then the causes underlying some of these errors are reviewed. This is followed by issues involved in reporting errors, limitations of KBR3, and plans for future enhancement.

KBR3 Tool Overview

The KBR3 tool is based on the KB-Reducer algorithm originally developed by Ginsberg (see Ginsberg, 1991). For KBR3, the basic algorithm was enhanced to handle rule bases containing equations. This enhanced algorithm is formally described in Williamson and Dahl, 1993.

Figure 1 provides a basic overview of the KBR3 analysis tools. A KB to be analyzed is first translated from its original source language into a application-neutral language tailored for KBR3. Then the "label" for each rule conclusion is computed. Each label is simply the set of input states (called "environments") that allow the conclusion to be deduced. These labels describe the KB functionality in terms of input/output pairs. Computing them essentially "compiles out" all the deductive paths of the original KB (often referred to as the "extension" of the KB).

The resulting labels are used by the "perform checks" module in detecting the errors types enumerated in the next section. Finally, the "Report Interface" module allows tool users to integrate the results in the context of the KBs original source language.

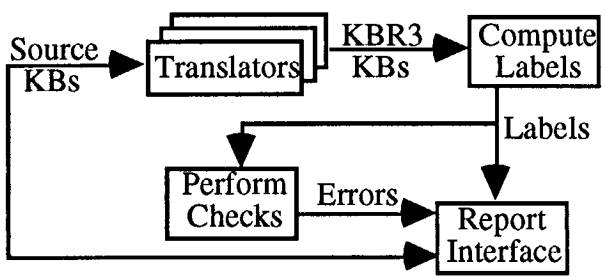

Figure 1 - KBR3 tool architecture

Context of tool usage

To understand the observations made in this paper, it is important to know that the KBR3 tools were used during KB maintenance and usually after some amount of conventional KB testing had been done. This conventional testing included syntax checking, checks for obvious constraint violations, and some manual testing of the

maintenance changes. In some cases, more thorough testing was done, such as regression testing or code inspection.

This paper focuses on KB maintenance because the major use of the KBR3 tools have thus far been by KB maintainers. Besides testing maintenance changes, the tools have been helpful in verifying a maintainer's understanding of KBs originally developed by others (see Dahl and Williamson 1992).

The reason why some conventional testing was done before employing the KBR3 tools was due to the large amount of time it takes to run the KBR3 analysis (see "Efficiency Limitations"). Thus the KBR3 analysis was reserved for finding the more elusive errors, such as order dependencies among rules. An incremental version of KBR3 may reduce these limitations (see "Future Directions").

Error Types

There are eleven types of errors detected by KBR3. Of these, inconsistency and redundancy are discussed in detail since they are the most general problem indicators and the most challenging for users to deal with. Since the other error types are relatively intuitive, this paper does not dwell on them. Formal definitions of all the error types can be found in Williamson and Dahl, 1993.

Inconsistent rules

Two rules are considered inconsistent if they assert logically contradictory conclusions under the same input states. Rules r1 and r2 are inconsistent in the following example:

```
r1: IF p THEN q
r2: IF s THEN NOT q
r3: IF p THEN s
```

Redundant rules

A rule is considered redundant if it can be removed from the knowledge base without effecting its functionality. This notion of redundancy is very general and has as special cases dead end rules, un-firable rules and subsumed rules.

Other error types

The other types of errors detected by KBR3 are: incompleteness, invalid conclusions, un-firable rules, rule subsumption, unused variables, un-computed variables, dead end rules, overlapping rules, and circular reasoning.

Errors vs. Their Causes

It is important to remember that there are many possible causes for each of the error types discussed in the last section. In Preece and Shinghal, 1992, it is noted that the distinction between these anomalies and their causes "has often been blurred in the literature". This blurring has promoted some inappropriate ideas on how to interpret errors and how to automate their correction.

For example, one popular strategy for error correction is to automatically remove all redundant rules. While this strategy preserves KB functionality, it could cover up any number of problems. Suppose during maintenance, a under-qualified rule is introduced. This new rule may unintentionally subsume other valid rules, causing them to be redundant and candidates for removal! The reverse strategy of removing the most general rule can also backfire if the more specific cases were mistakenly added to a valid KB already containing the general rule (of course, the general rule was never redundant anyway).

Another example of this blurring is making the assumption that a logical inconsistency found in a KB is due to an inconsistent mental model on the part of the knowledge engineer. For instance, in the following example, one strategy of automatic error correction suggests the most general rule, r2, is at fault and should be specialized. However, it is possible that the error is just a typo and r1's antecedent should have been: `NOT P AND Q`.

```
r1: IF P AND Q THEN R
r2: IF P THEN  NOT R
```

The next two sections survey the causes of inconsistencies and redundancies uncovered with the KBR3 analysis tool.

Causes of Inconsistency

This section describes the seven causes of inconsistency that were uncovered by using the KBR3 tools on ESDS KBs: order-dependent rules, inconsistent KB specifications, inputs correctly assumed to be unrealistic, inputs improperly assumed unrealistic, under-qualified rules, overly complex rules with overlapping conditions, and typos or cut and paste errors.

Order-dependent rules

One of the often cited benefits of rule-based system is the modularity of rules. However, this modularity often breaks down in practice due to unforeseen rule inter-actions. One type of rule inter-action that hinders maintenance occurs when rules deducing values for the same attribute are written in an order-dependent fashion. As reported in Dahl and Williamson, 1992, order dependent rules are maintenance traps since they leave knowledge implicit (e.g., it is not stated which rules are dependent on which others and why).

Unfortunately, it is difficult in practice to prevent the introduction of order-dependent rules. This is because

the inference engine that processes the rules has a fixed strategy which is usually understood and (often unwittingly) exploited when writing rules. Order dependent rules may not be detected by running KB test cases since these rules may not cause the KB to produce erroneous answers.

Fortunately, the inconsistency detection of the KBR3 analysis tool can uncover order-dependent rules. Whether this is done depends on an option set when running the KB to KBR3 language translators shown in figure 1. To suppress detection of order dependency in the example below, the translator conjoins the negation of the antecedent of rule r1 to the antecedent of rule r2, resulting in a new antecedent for rule r2 of: `NOT (x = 1) AND y = 1`. Without this option set, KBR3 would find rule r2 inconsistent with rule r1.

```
r1: IF x=1 AND y=1 THEN a=10
r2: IF         y=1 THEN a=20
r3: DEFAULT VALUE OF a=30
```

As suggested by the syntax in the example above, rule r3 specifies default information and can never be considered inconsistent with other rules. The KBR3 language provides a convenient way for explicitly representing default mechanisms via a default rule that names all other rules (in this case r1 and r2) whose conditions must be false for the default rule to fire.

With this arrangement, explicit order dependencies are ignored while implicit ones are optionally uncovered. If, as sometimes is the case, an order dependency detected by KBR3 is caused by a rule acting to represent default information that can not be explicitly identified as such in the source language, then the resulting inconsistency error can be marked to be ignored by KBR3 in the future. For instance, rule r2 in the previous example may act as a more specific default which should be considered before r3.

One may observe that notations for specifying defaults are logically unnecessary since defaults can always be expressed as the negation of the disjunction of all the other cases. This, however, is a maintenance problem, since the conditions for defaults must be kept in sync with the non-default cases and amounts to an error-prone form of redundant information. In fact, the existence of this maintenance problem is probably the best indicator of whether a rule should be considered as representing default information.

Using KBR3 to detect rule order dependencies has turned out to be a very valuable tool for uncovering knowledge that was left implicit. It is a unique test for KB quality and maintainability which is not addressed by other conventional testing methods. A majority of the inconsistencies found thus far have been of this type.

Inconsistent KB specifications

The most obvious cause of a logical inconsistency where the specifications on which the KB is based are themselves logically inconsistent (by "specification", we mean either formal documentation of mental models of the domain expert or knowledge engineer).

Surprisingly, none of the inconsistencies uncovered so far by the KBR3 tool are attributable to inconsistent specifications. This is probably because KBR3 has thus far only been used for checking maintenance changes that have already been tested in other ways (e.g, regression testing, manual testing). Of course, this is not to say that such inconsistencies will never be uncovered.

This finding does, however, emphasize the importance of distinguishing between error types, as defined by their syntactical form, and the underlying causes for their introduction into a KB.

Inputs correctly assumed to be unrealistic

A logical inconsistency detected by KBR3 can be caused when a set of inputs is allowed by the KB that would never in practice be entered by any KB user. A simple example might be:

```
r1: IF airplane-model = "747" THEN
          travel-range = "long"
r2: IF number-of-engines = 1  THEN
          travel-range = "short"
```

KBR3 would treat rules r1 and r2 as inconsistent, unless the KB explicitly disallowed consideration of a single engine 747. This could be done with the explicit constraint: `NOT (airplane-model = "747" AND number-of-engines <> 4)`.

Apparently this type of inconsistency was prevalent in the KBs used to test the initial version of the KB Reducer algorithm that was used at Bell Labs. In our experience of using KBR3 on ESDS KBs, we have found relatively few inconsistencies that were due to such under-constrained combinations of inputs.

Inputs improperly assumed unrealistic

Another more serious cause of inconsistencies found by KBR3 is like the last, except that the KB builders incorrectly assumed an input set would be viewed as "unrealistic" by users. The KBR3 tool found such an inconsistency in the first real KB it analyzed. A simplification of the rules involved was:

```
r1: IF part-is-machined
  THEN NOT use-transparent-material
r2: IF part-must-be-seen-through
  THEN     use-transparent-material
```

The problem was actually in the definition of machined part. The domain expert did not expect that users would

```
r1: IF ((P AND NOT (v1 IN-LIST L)
         AND NOT (v2 IN-LIST L))
        OR Q)
       AND -50 <= X-max <= 350
       AND -50 <= X-min <= 350
       AND Y  = y1
       AND (Z IN-LIST (z1, z2) OR
            (Z = z1 AND NOT R))
   THEN Answer  = answer-1
```

```
r2: IF (Q OR
         (P AND NOT (v1 IN-LIST L)))
       AND -65 <= X-max <= 400
       AND -65 <= X-min <= 400
       AND Y  = y1
       AND Z = z1
   THEN Answer  = answer-2
```

Figure 2 - Overlapping Complex Rule Conditions

consider windows as machined parts because of the modest cutting and drilling involved. The root of the problem was an imprecise definition of what constituted a machined part.

Although we have so far only found a few inconsistencies of this type, the ability to detect them is a valuable contribution of the KBR3 tool.

Under qualified rules

Another cause of inconsistencies found by KBR3 is the familiar problem of under-qualified rule conditions. One such error had the form:

```
r1: IF 100 < x < 500 AND P
          THEN y = 20
r2: IF 125 < x < 500 AND P AND Q
          THEN y = 50
```

The problem was that r1 left out the condition: `NOT Q`. Thus while the rules seemed to cover overlapping conditions, the conditions should have been mutually exclusive. In the case alluded to above, the error may have not been found with conventional testing since the error's only effect was to sometimes provide overly safe recommendations.

Overly complex rules with overlapping conditions

Some knowledge engineers (and even some domain experts) "enjoy" writing complex boolean expressions. Unfortunately they rarely enjoy reading them. Such expression also tend to contain errors, especially when taken in combination. KBR3 has uncovered such errors, usually as inconsistencies among rules. Figure 2 shows the form of one case encountered.

Typos and cut and paste errors

It is possible to imagine all kinds of logical errors that can be caused by editing mistakes. In practice, not many errors found by KBR3 were attributed to editing mistakes. Most that were so attributed were not identified as inconsistencies, but as redundancies.

Causes of Redundancy

All but two of the causes of inconsistencies covered in the last section can in theory also cause redundancy. The cause called "overly complex rules with overlapping conditions" does not apply because rules with overlapping conditions and identical actions are not technically redundant (there is a separate test for this). Obviously, the other cause that can not apply to redundancy is "Inconsistent KB Specifications".

From the KBs analyzed, the causes of inconsistency that were actually found to also cause redundancy were: "Order-dependent rules", "Under qualified rules", and "Typos and cut and paste errors". The two causes involving unrealistic inputs were not seen, although there is no reason to think they never will. There were always less redundancies reported than inconsistencies.

While inconsistency always involves rules that conclude different values for the same input, redundancy involves more than just rules concluding the same values for the same input. Since a rule is redundant if it can be removed without affecting KB behavior, redundancy also involves dead code in the form of un-firable rules or dead-end rules (i.e. rules not contributing to KB outputs). For this reason, typos in attribute values were a common cause of redundancy, since they often produce un-firable rules.

Other causes found for redundancy include:

Intentional mirroring of redundant knowledge sources

There were several cases where a KB intentionally had rules subsumed by others because the rules directly represented knowledge obtained from different sources and direct traceability to those sources was an important validation goal.

Intentional dead code

Some KBs had a lot of dead code (un-firable or dead-end rules) which was intentionally "dead", either because it was part of an in-work section of the KB or was temporarily disabled.

```
r1: IF x=1 THEN y=10                    r4: IF y=10 THEN z=100
r2: IF x=1 THEN y=20                    r5: IF y=20 THEN z=200
r3: IF x=2 THEN y=20          r6: IF y=20 AND x <> 1 THEN z=300
```

Figure 3 - Example of Error Propagation

Unintentional Dead Code

This form of dead code was often created in the process of restructuring a KB. Sometimes such code was initially left to fall back on in case the new code did not work out. It is helpful to identify this code since it can be a stumbling block for future maintenance.

Invalid assumptions by KBR3

One cause of redundancy reported by KBR3 had nothing to do with the KBs analyzed but rather was an invalid assumption made about the inference engine that the KB was written for. The KBR3 analysis mistakenly assumed string comparisons should be case sensitive. This cause of "redundancy" is worth noting since knowledge representation languages and inference schemes used by real KBs are often rich and not fully documented.

Issues in Error Reporting

Two problems in error reporting were experienced, both involve single errors which generate multiple error messages.

Error propagation

Under some circumstances, errors can be propagated from the rules generating them to other rules that depend on the former rules. Take for example, the set of rules shown in figure 3.

In figure 3, the inconsistency between r1 and r2 would be propagated to rules r4 and r5. Note that the error will not propagate to rules r4 and r6 because the condition "x <> 1" prevents r2 contributing to r6.

To partially alleviate the problem having users wade through propagated errors, the errors are sorted by rule level, so that propagated errors are always reported after their primary errors. In practice, this approach is usually adequate but incurs the risk of missing primary errors that exist at the same level as a sea of propagated errors. We are looking for more effective approaches that would let us detect and remove propagated errors. It is possible that the approach taken in Zlatareva, 1992, can handle this problem.

Number of actual vs. reported errors

Beyond the extra errors reported due to error propagation, the KBR3 algorithm will report a single logical inconsistency many times, once for each assignment of values to non-input attributes that cause the inconsistency to manifest itself.

```
r1: IF A=B THEN R
r2: IF A=B THEN ~R
```

In the above example, if attributes A and B were both not inputs and could each take on 10 values, then KBR3 would report 100 errors (one for each pair of A and B values). Fortunately, these individual error reports are grouped together, allowing the user to skip over most of them. This practice does have a slight drawback since the user may not discover that some rules have more than one inconsistency between them (e.g., suppose r1 and r2 both had an antecedent of: `A=B OR C=D`)

KBR3 Limitations and Work-Arounds

The KBR3 approach has a couple of inherent limitations for which we have found practical work-arounds.

Problems related to undecidability

Because KBR3 analyzes KBs containing algebraic equations, it cannot be guaranteed to find all errors, and in fact may at times report erroneous errors. For example, since the present version of KBR3 does not have any knowledge of trigonometric identities, it won't recognize that rules r1 and r2 are inconsistent. Likewise it will erroneously report rules r3 and r4 as inconsistent.

```
r1: IF sin(90 + x) > 0 THEN P
r2: IF cos(x) > 0 THEN ~P
r3: IF x = 1 THEN y=sin(90+x)
r4: IF x = 1 THEN y=cos(x)
```

The approach we have taken to this problem is to add a limited suite of algebraic rules which are expanded as the need arises. Fortunately, the KBs we are now analyzing only contain very simple algebraic expressions and the current set of rules have proven adequate.

Efficiency limitations

We have found it computationaly prohibitive to run the KBR3 analysis on many of our large KBs. We have also found that size of a KB alone is not a good indicator of how long the analysis may take. For example, two small KBs were analyzed, both with roughly 150 rules. One took only 5 minutes, while the other took 120 hours! What appears more important than KB size is the amount

of rule inter-connectivity, length of inference chains, and the number of non-input attributes which have a large number of explicitly stated values and which are also referenced in rule conclusions.

The costlier KBs can still be partially analyzed. One approach we employ is to use tools that derive a subset of a KB that supports KBR3 analysis of a requested set of attributes or rules. While an analysis is in progress, it is also possible to request obviously costly rules (and their dependants) to be skipped.

Future Directions

There are many potential enhancements which could be made to the KBR3 tool suite. Enhancements we are considering include:

Incremental analysis

The efficiency limitations mentioned in the last section motivate ongoing research into an incremental version of the KBR3 analysis. Incremental analysis will make it unnecessary to re-analyze portions of a KB unaffected by KB maintenance changes. This should allow the KBR3 tool to be used more frequently.

Error interpretation/correction support

The variety of causes we encountered for errors detected by KBR3 tools have shown that strategies for fixing errors must depend on the underlying causes of errors and not just the syntactical form of the errors. Therefore, we are not currently trying to automate error correction. Instead, we are working to provide information relevant to error interpretation via a rich user interface that incorporates hyper-text links.

Test case generation

We are looking into using the results of KBR3 analysis to augment regression test suites with tests that cover newly added KB input-output behavior.

Integration/Redundancy management tools

We are looking into tools based on KBR3 analysis which automatically detect redundancies across KBs as well as general frameworks to manage replicated knowledge. Such tools will probably not be practical without an incremental version of the KBR3 analysis.

Conclusions

Verification tools such as KBR3 are proving to be effective in detecting otherwise hard-to-find anomalies in knowledge bases, despite the computational cost of performing the analysis. Moreover, the analysis can uncover potential maintenance traps that are usually not found by running the KB, including hidden dependencies on rule ordering and under-constrained inputs. Interpreting the results of the analysis can sometimes be challenging since there are many potential causes for the errors detected, especially in the case of the general errors of inconsistency and redundancy. Due to the computational cost and time required by KBR3, it makes sense to first screen out obvious errors with less expensive tools or methods. An incremental version of KBR3 could reduce the need for this pre-screening.

References.

Dahl, M. 1993. ESDS: Materials Technology Knowledge Bases Supporting Design of Boeing Jetliners. To appear in the *proceedings of the 1993 conference of Innovative Applications of Artificial Intelligence (IAAI-93).*

Dahl, M. and Williamson. K. 1992. A Verification Strategy for Long-Term Maintenance of Large Rule-Based Systems. In *Workshop notes for the AAAI-92 Workshop on Verification and Validation of Expert Systems.*

Ginsberg, A. 1991. Knowledge-Base Reduction: A New Approach to Checking Knowledge Bases For Inconsistency & Redundancy. In *Validating and Verifying Knowledge-Based Systems.* IEEE Computer Science Press.

Preece, A and Shinghal, R. 1992. Analysis of Verification Methods for Expert Systems. In *Workshop notes for the AAAI-92 Workshop on Verification and Validation of Expert Systems.*

Stackowicz, R. 1991. *Validation of Knowledge Based Systems.* Lockheed Missile & Space Co. Technical Report #RL-TR-91-361.

Williamson, K. and Dahl, M. 1993. Knowledge-Based Reduction for Rule Bases Containing Equations. To appear in *Workshop notes for the AAAI-93 Workshop on Verification and Validation of Expert Systems.*

Zlatareva, N. 1992. Knowledge Refinement via Knowledge Validation. In *Workshop notes for the AAAI-92 Workshop on Verification and Validation of Expert Systems.*