WHERE DO WE GO FROM HERE?
Applying KBS V&V Technology

R.F. Gamble
A.V. Pai
University of Tulsa
Tulsa, OK, USA

R.T. Plant
University of Miami
Miami, FL, USA

Introduction

As the field of verification and validation for knowledge-based systems (KBSs) has matured, much information, technology, and theory has become available. Though not all of the problems with respect to KBSs have been solved, many have been identified with solutions that can be used in an analogous manner in situations where the application is not necessarily a traditional KBS. The value of the research that the V&V community has conducted increases substantially as it is shown to be wider reaching than traditional KBSs. In this research note, we apply KBS V&V research to active databases where we show that similar anomalies can occur.

The "active" component in an active database (ADB) consists of rules that execute as a result of database accesses and updates. These rules and their inference mechanism can be abstracted from the database to form the *rule processing component.* We show how the rule structures of various ADB rule processing components can be isolated and converted into a consistent intermediate form similar to a KBS rule. We borrow from verification research in KBSs to detect anomalies that may be present in ADBs. Because the incorporation of active components in databases is becoming standard, this work illustrates the danger these anomalies can pose. Fortunately, tools from KBS verification already exist to aid ADB verification.

Active Databases

The dynamic properties of an active database, that cause specified actions to occur automatically when certain conditions in the database arise, can be extremely useful to programmers. For example, consider an inventory database with a large number of products where each contains a quantity on hand, threshold, and reorder flag. In a traditional database, as products are sold, the quantity on hand is continually updated. An ADB goes one step further by monitoring the quantity on hand such that if it falls below the threshold, the reorder flag is set. A *trigger* or *rule*, containing an event, condition, and action (E-C-A) is placed in the "quantity on hand" field. When its associated event (an update to the quantity on hand) occurs, the condition of the trigger is evaluated. If the condition evaluates to true, the action is executed and the reorder flag is set accordingly. The ability to detect events and evaluate/execute rules over a large amount of data make active databases more flexible than traditional database systems [1, 2, 3]. The ability to manipulate data with rules indicates the need for a sound rule processing component to ensure the overall integrity.

A database trigger in Oracle is implicitly executed when an INSERT, DELETE, or UPDATE event is issued against the table for which the trigger is defined. A trigger is able to fire only when it is enabled, at which time it can also fire other enabled triggers, producing a cascading effect. The basic format of a database trigger consists of a *triggering event*, a *trigger restriction*, and a *trigger action* and is depicted in Figure 1. A complete formal definition is provided in [4, 5]. A triggering event in Oracle is the SQL statement that causes the trigger to fire. A trigger restriction specifies a Boolean expression that must be TRUE for the trigger to fire. A trigger action is the procedure that contains the code to be executed when the trigger restriction evaluates to true.

Triggering Statement:	AFTER UPDATE OF column name ON table name
Trigger Restriction (Optional):	WHEN (new.column nameA < new.column nameB)
Trigger Action:	FOR EACH ROW DECLARE variable and variable data type BEGIN IF... THEN...END IF... END.

Figure 1: Abstracted Oracle Trigger [15]

As many as twelve triggers can be associated with a single table. Multiple triggers can be fired. Conflict resolution strategies have been defined to manage multiple trigger firings [5].

KBS Verification and Validation Applications

Potential KBS Anomalies

KBS verification includes examining the structure and semantics of a knowledge base. Primary verification tools generally analyze the knowledge base for anomalies, which may be logical, epistemological, or semantic, and often fall into the following categories [6, 7]: *redundancy, conflicting rules, circularity,* and *errors of omission.* These categories of anomalies and methods of their detection have already been widely researched for various types of KBSs, including rule-based and hybrid (object-oriented and rule-based) systems [6, 8, 9, 10, 11, 13, 14,15, 16].

Structural Similarities with ADBs

The logical form of a knowledge-based system rule and an active database rule are somewhat similar. However, the rule processing within ADBs differs from the rule processing in KBSs. A KBS typically uses a match-select-execute cycle in which the knowledge base holds the rules, the working memory holds the available facts, and the inference engine deduces new information by comparing the contents of working memory to the rules and executing some subset of matching rules. An ADB, on the other hand, uses coupling mode information that indicates when the conditions and actions of rules are to be evaluated and executed relative to a database event.

In order to analyze the ADB rules such that verification is facilitated, we must transform them to a consistent generic representation as follows:

Given the Oracle rule:

```
Rule 5
CREATE TRIGGER rchain1
AFTER UPDATE OF quantity
ON inv
FOR EACH ROW
WHEN (((new.QUANTITY -
        old.DEFECTED_QTY) <=
                    old.THRESHOLD)
      AND (old.status =
                'UNPREPARED'))
BEGIN
      UPDATE stock
      SET status = 'MIN'
      WHERE itemNo = :old.itemNo;
END
```

we identify the Event, Condition and Action (E-C-A) parts as:

Event: UPDATE OF quantity ON inv
Condition: ((new.QUANTITY -
old.DEFECTED_QTY) <=
old.THRESHOLD)
AND (old.status =
'UNPREPARED')
Action: UPDATE stock SET status = 'MIN'
WHERE itemNo = :old.itemNo;

These parts are converted into predicates and represented in a generic E-C-A format where E1 and E7 represent events.

Event: E1
Condition: P(x, y, z) AND Q(w)
Action: E7 AND T(v)

The next transformation is to a knowledge base Condition-Action format.

Condition: E1 AND P(x, y, z) AND Q(w)
Action: E7 AND T(v)

The intermediate representation given above serves as a generic language in which to convert different ADB representations of rules into a consistent format for analysis. By abstracting away the particular syntax of an ADB rule set we can efficiently isolate potential anomalies.

Identifying Potential ADB Rule Anomalies

In this section, we briefly show how the generic representation allows for direct anomaly detection using KBS verification techniques. For example, one method involves the construction of an event dependency tree. In order to illustrate potential anomalies, a rule set in Oracle 7 ADB rule syntax was created that contains anomalies for redundancy, incompleteness, and inconsistency [12]. The portion of the Oracle and KBS rule sets that contain redundancies are described in Table 1.

ORACLE ADB RULE SET	KBS RULE SET
RULE 1 CREATE TRIGGER rule1 AFTER UPDATE OF quantity ON inv FOR EACH ROW WHEN (((new.QUANTITY - old.DEFECTED_QTY) <= old.THRESHOLD) AND (old.status = 'UNPREPARED')) BEGIN reorder(:old.itemNo); END;	R1: E1,P(x,y,z), Q(w) → E2
RULE 2 CREATE TRIGGER subsumption AFTER UPDATE OF quantity ON inv FOR EACH ROW WHEN (((new.QUANTITY - old.DEFECTED_QTY) <= old.THRESHOLD) AND (old.status = 'UNPREPARED') AND (old.ON_RECALL_LIST = 'TRUE')) BEGIN reorder(:old.itemNo); END;	R2: E1,P(x,y,z),Q(w), R(v) →E2
RULE 3 CREATE TRIGGER duplication AFTER UPDATE OF quantity ON inv FOR EACH ROW WHEN (((new.QUANTITY - old.DEFECTED_QTY) <= old.THRESHOLD) AND (old.status = 'UNPREPARED')) BEGIN reorder(:old.itemNo); END;	R3: E1,P(x,y,z), Q(w) → E2
RULE 4 CREATE TRIGGER unnec_if AFTER UPDATE OF quantity ON inv FOR EACH ROW WHEN (((new.QUANTITY - old.DEFECTED_QTY) <= old.THRESHOLD) AND (old.status = 'PREPARED')) BEGIN reorder(:old.itemNo); END;	R4: E1,P(x,y,z), ¬Q(w) → E2
RULE 5 CREATE TRIGGER rchain1 AFTER UPDATE OF quantity ON inv FOR EACH ROW WHEN (((new.QUANTITY - old.DEFECTED_QTY) <= old.THRESHOLD) AND (old.status = 'UNPREPARED')) BEGIN UPDATE stock SET status = 'MIN' WHERE itemNo = :old.itemNo; END;	R5: E1,P(x,y,z), Q(w) → E7, T(v)
RULE 6 CREATE TRIGGER rchain2 AFTER UPDATE OF status ON stock FOR EACH ROW WHEN (new.status = 'MIN') BEGIN reorder(:old.itemNo); END;	R6: E7,T(x),S(y) → E2

Table 1: An Example Rule Set

The different types of redundancy can be better seen when the KBS rules are grouped together in their redundant configurations as in Table 2. The results of actually testing the Oracle rules are described in the last column. As the reader can see, there are no built-in database safeguards for the anomalies already known to V&V community.

Redundancy Type	Rule Groups	Result
Subsumption	R1: E1,P(x,y,z), Q(w) → E2 R2: E1,P(x,y,z), Q(w), R(v) → E2	Both rules execute The qty_reorder attribute of the On_Reorder table updated twice
Reduceable	R1: E1,P(x,y,z), Q(w) → E2 R4: E1,P(x,y,z), ¬Q(w) → E2	Both rules execute The qty_reorder attribute of the On_Reorder table updated twice
Duplication	R1: E1,P(x,y,z), Q(w) → E2 R3: E1,P(x,y,z), Q(w) → E2	Both rules execute The qty_reorder attribute of the On_Reorder table updated for all attributes regardless of the restricting conditions on the status of the Inv table
Indirect	R1: E1,P(x,y,z), Q(w) → E2 R5: E1,P(x,y,z), Q(w) → E7, T(v) R6: E7,T(x),S(y) → E2	All three rules execute The qty_reorder attribute of the On_Reorder table updated twice.

Table 2: Anomaly Rule Groups

While a simple event dependency graph can isolate redundancy anomalies, further analysis of the rules is usually required prior to confirming a problem. For example, if the rule R0, shown in Figure 2, were a part of the rule set in Table 1, it would be seen as potentially redundant with respect to events E1 and E2. However, upon comparing this rule with the other rules that are triggered by E1 and reach E2, it is concluded that this rule is not redundant with the others since the LHSs of R0 and the other rules leading to event E2 are not problematic.

RULE 0

```
CREATE TRIGGER notRedundant
AFTER UPDATE OF quantity
ON inv
FOR EACH ROW
WHEN (new.ItemNo = '10A')
BEGIN reorder(:old.itemNo); END;
```

Figure 2: Rule illustrating additional analysis after anomaly detection

5. Conclusion

KBS verification and validation techniques have greatly matured to the point where they can be applied to analogous types of computation in different domains. We have shown that these techniques are valuable for detecting anomalies within the rule processing component of active databases, where no safeguards against such anomalies were considered as part of including an "active" component in a database.

References

1. Hanson, E.N. (1991), The Design and Implementation of the Ariel Active Database Rule System, Technical Report UF-CIS-018-92, Department of Computer and Information Sciences, University of Florida.

2. Stonebraker, M., Hanson, E., and Potaminos. S. (1988), The POSTGRES Rule Manager, *IEEE Transactions on Software Engineering*, 14(7):897-907.

3. Dayal, U. (1988), Active Database Management Systems, *Proc. of Conf. of Data and Knowledge Bases.* Jerusalem.

4. Hanson, E.N., and Widom.J. (1993), Rule processing in active database systems, *International Journal of Expert Systems*, 6(1):83-119.

5. Oracle (1992), Oracle Corporation, Redwood Shores, CA 94065. Oracle 7 Server Manuals, Release 7.1.

6. Gamble, R.F. and Shaft T.M. (1996), Eliminating Redundancy, inconsistency, and incompleteness in rule-based systems, *International Journal of Software Engineering and Knowledge Engineering,* Vol. 7, No. 4.

7. Murrell, S., and Plant, R.T. (1995), Decision Tables: Formalisation, Validation and Verification, *The Journal of Software Testing, Verification and Reliability,* Vol 5, (107-132).

8. Murrell, S. and Plant, R.T. (1997), A Survey of Tools for the Verification and Validation of KBS: 1985-95, *Decision Support Systems*, 21:4.

9. Plant, R.T. (1994), Validation and Verification of Knowledge-Based Systems. Workshop Notes AAAI. 1994, Seattle, WA.

10. Gamble, R. and Landauer, C. (1995), Validation and Verification of Knowledge-Based Systems. Workshop Notes IJCAI-95, Montreal, Quebec. Canada.

11. Mukherjee, R., Gamble, R.F., and Parkinson. J.A., (1997), Classifying and Detecting Anomalies in Hybrid Knowledge Based Systems, *Decision Support Systems,* 21:4.

12. Pai, A.V. (1995), Verifying the rule processing components of active databases, M.S. Thesis, Dept. of Mathematical and Computer Sciences, University of Tulsa.

13. O'Leary. D.E. (1989), Verification of frame and semantic network knowledge bases, *AAAI-89 Workshop on Knowledge Acquisition for KBSs.*

14. Preece,A.D., Shinghal, R., and Batarek, A. (1992), Verifying expert systems: A logical framework and practical tool, *Expert Systems with Applications*, 5:421-436.

15. O'Keefe, R. and O'Leary, D.E. (1993), Expert system verification and validation: A survey and tutorial, *Artificial Intelligence Review*, 7:3 - 42.

16. Schmolze, J. and Vermesan, A. (1996), Validation and Verification of Knowledge-Based Systems. Workshop Notes AAAI-96, Portland, OR.