On the Verification of Expert Systems: A case study on the Power Systems Area

Jorge Santos, Zita Vale and Carlos Ramos
GECAD - Knowledge Engineering and Decision Support Group
Institute of Engineering - Polytechnic of Porto
Porto, Portugal
{ajs,zav,csr}@isep.ipp.pt

Abstract—

The Verification and Validation (V&V) process states if the software requirements specifications have been correctly and completely fulfilled.

The methodologies proposed in software engineering showed to be inadequate for Knowledge Based Systems (KBS) validation and verification, since KBS present some particular characteristics.

Designing KBS for dynamic environments requires the consideration of Temporal knowledge Reasoning and Representation (TRR) issues. Although humans present a natural ability to deal with knowledge about time and events, the codification and use of such knowledge in information systems still pose many problems. Hence, the development of applications strongly based on temporal reasoning remains an hard and complex task. Furthermore, albeit the last significant developments in TRR area, there is still a considerable gap for its successful use in practical applications.

VERITAS is an automatic tool developed for KBS verification which is able to detect a large number of knowledge anomalies. It addresses many relevant aspects considered in real applications, like the usage of rule triggering selection mechanisms and temporal reasoning. VERITAS is currently in development and is being tested with SPARSE, a KBS used in the Portuguese Transmission Network (REN) for incident analysis and power restoration. In this paper some solutions are proposed for still open issues on Verification of KBS applied in critical domains.

I. Introduction

Knowledge has been recognized as distinguish factor and added value between the organizations, leading to growing efforts in knowledge maintenance process. Knowledge Engineering (KE) takes a paramount role in this scenario while Knowledge Based Systems are one of the most effective and reliable tools in use for KE.

The Verification and Validation process is part of a wider process denominated Knowledge Maintenance[7] in which an enterprise systematically gathers, organizes, shares, and analyzes knowledge to accomplish its goals and mission.

Many authors argued that the correct and efficient performance of any piece of software must be guaranteed through the Verification and Validation (V&V) process and it becomes obvious that KBS (Knowledge Based Systems) should undergo the same evaluation process.

The methodologies proposed in software engineering showed to be inadequate for Knowledge Based Systems (KBS) validation, since KBS present some particular characteristics[5]. Namely, KBS needs to deal with uncertainty and incompleteness; the domains modelled normally do not underline physical models; it is not rare that KBS have the ability to learn and improve the KB allowing a dynamical behavior and in most domains of expertise, there is no concept of *right* results but *acceptable* ones.

It is known that knowledge maintenance is an essential issue for the success of the KBS since it assures the consistency of the knowledge base (KB) after each modification in order to avoid the assertion of knowledge inconsistencies.

In last decades Knowledge Based Systems became a common tool in a large number of Power Systems Control Centres (CC). In fact, the quantity, diversity and complexity of KBS increased significantly leading to important changes in KBS structure. Designing KBS for dynamic environments requires the consideration of TRR (Temporal Reasoning and Representation) issues. Although humans present a natural ability to deal with knowledge about time and events, the codification and use of such knowledge in information systems still pose many problems. Hence, the development of applications strongly based on temporal reasoning remains an hard and complex task. Furthermore, in despite of the last significant developments in TRR area, there is still a considerable gap for its successful use in practical applications.

Besides the facets of software certification and maintenance previously referred, the systematic use of formal V&V techniques is also a key for making end-users more confident about KBS, especially when critical applications are considered. In Control Centres domain the V&V process intends to assure the reliability of the installed applications, even under incident conditions.

VERITAS is an automatic tool developed for KBS verification, it is currently in development and is being tested with a KBS developed for the Portuguese Transmission Network (REN[1]) in the incident analysis and power restoration, named SPARSE [18]. VERITAS tool performs KB structural analysis allowing knowledge anomalies detection. Originally, VERITAS used a non temporal KB verification approach. Al-

[1] www.ren.pt

though it proved to be very efficient in other KBS verification in SPARSE case some important limitations were detected.

This paper addresses the Validation and Verification of Knowledge-Based Systems in general, focussing particularly on the SPARSE's V&V process, reporting VERITAS last improvements. Some solutions are proposed for still open issues on Verification of KBS applied in critical domains, namely, we propose an approach for the verification of KBS temporal properties based on ontologies combination.

The rest of the paper is organized as follows. Section II provides a short overview of the state-of-art of V&V and its most important concepts and techniques. Section **??** presents VERITAS, an automatic verification tool. Special emphasis is given to the tool architecture and to the method used in anomaly detection. Finally, in section V, we present some conclusions and topics for future work.

II. Related Work

The problem of Verification and Validation appears when there is a need to assure that some model (solution) correctly addresses the problem through the adequate techniques and methodology in order to provide the desired results. In the scope of this work, Validation and Verification will be referred as two complementary processes, both fundamental for KBS end-user acceptance. The following definitions will be used for the rest of this paper.

Definition II.1 (Validation) Validation means building the right system *[1]. The purpose of validation is to assure that a KBS will provide solutions with similar (or higher if possible) confidence level as the one provided by domain experts.*

Validation is then based on tests, desirably in the real environment and under real circumstances. During these tests, the KBS is considered as a *black box*, meaning that, only the input and the output are really considered important.

Definition II.2 (Verification) Verification means building the system right *[1]. The purpose of verification is to assure that a KBS has been correctly designed and implemented and does not contain technical errors.*

During the verification process the interior of the KBS is examined in order to find any possible errors, this approach is also called *crystal box*.

Definition II.3 (Verification & Validation) *The Verification and Validation process allows to determine if the requirements have been correctly and completely fulfilled in order to assure the system reliability, safety, quality and efficiency.*

More synthetically, it can be said that the V&V process is to *build the right system right* [11].

In the last decades several techniques were proposed for validation and verification of knowledge based systems, like, inspection, formal proof, cross-reference verification or empirical tests, the efficiency of these techniques strongly depends on the existence of test cases or in the degree of formalization used on the specifications. One of the most used techniques is static verification, that consists of sets of logical tests executed in order to detect possible knowledge anomalies. Regarding that an anomaly is a symptom of one (or multiple) possible error(s). Notice that an anomaly does not necessarily denotes an error [13].

Knowledge bases are drawn as a result of a knowledge analysis/elicitation process, including, for example, interviews with experts or the study of documents such as codes of practice and legal texts, or analysis of typical sample cases. Hence, the KB should reflect the nature of this process. Consequently, some anomalies are desirable and intentionally inserted in KB. For instance, redundancy on the documentary sources will led to redundant KB.

Some well known V&V tools used different techniques to detect anomalies. The KB-REDUCER [4] system represents rules in logical form, then it computes for each hypothesis the corresponding labels, detecting the anomalies during the labelling process. Meaning, that each literal in the rule LHS (Left Hand Side) is replaced by the set of conditions that allows to infer it. This process finishes when all formulas became grounded. The COVER [12] works in a similar fashion using the ATMS (Assumption Truth Maintaining System) approach [6] and graph theory, allowing to detect a large number of anomalies. The COVADIS [15] successfully explored the relation between input and output sets.

The systems ESC [2], RCP [17] and CHECK [9] and more recently PROLOGA [19] used decision table methods for verification purposes. This approach proved to be quite interesting, specially when the systems to be verified also used decision tables as representation support. These systems major advantage is that tracing reasoning path becomes quite clear, while, the major problem is the lack of solutions for verifying long reasoning inference chains.

Some authors studied the applicability of Petri nets [10, 8] to represent the rule base and to detect the knowledge inconsistencies. More recently colored Petri nets were used [21]. Although specific knowledge representations provide higher efficiency while used to perform some verification tests, arguably all of them could be successful converted to production rules.

Rule based systems are still the representation more often used in the development of KBS. The scientific community has studied deeply these systems. At the moment there is an assortment of V&V techniques that allow the detection of many anomalies in systems that use this kind of representation.

III. The Verification Problem

The anomaly detection relies on the computation of all possible inference chains (expansions) that could be entailed during the reasoning process. Later, some logical tests are performed in order to detect if any constraints violation takes place.

SPARSE presents some features (see Fig.1) that made the verification work complex. These features demand the use of more complex techniques during anomaly detection and introduce significant changes in the number and type of anomalies to detect.

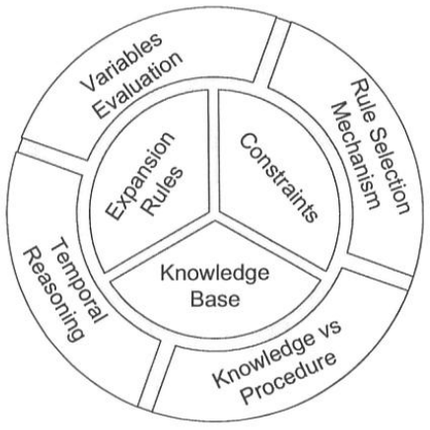

Fig. 1. The verification problem

A. Rule triggering selection mechanism

In what concerns SPARSE, this mechanism was implemented using both meta-rules and the inference engine. When a *message* arrives, some rules are selected and scheduled in order to be later triggered and tested. In what concerns verification work, this mechanism not only avoids some run-time errors (for instance circular chains) but also introduces another complexity axis to the verification. Thus, this mechanism constrains the existence of inference chains and also the order that they would be generated. For instance, during system execution, the inference engine could be able to assure that shortcuts (specialists rules) would be preferred over generic rules.

B. Variables evaluation

In order to obtain comprehensive and correct results during the verification process, the evaluation of the variables present in the rules is crucial, especially in what concerns temporal variables, i.e. the ones that represent temporal concepts. Notice that during anomaly detection (this type of verification is also called static verification) it is not possible to predict the exact value that a variable will have.

1) Temporal reasoning: This issue received large attention from scientific community in last two decades (surveys covering this issue can be found in [20, 3]). Despite the fact that *time* is ubiquitous in the society and the natural ability that human beings show dealing with it, a widespread representation and usage in the Artificial Intelligence domain remains scarce due to many philosophical and technical obstacles. SPARSE is an *alarm processing application* and its most difficult task is reasoning about events, thus, it is necessary to deal with time intervals (e.g. temporal windows of validity), points (e.g. events occurrence), alarms order, duration and the presence or/and absence of data (e.g. messages lost in the collection/transmission system).

C. Knowledge versus Procedure

Languages like Prolog provide powerful features for knowledge representation (in the declarative way) but they are also suited to describe procedures, so, sometimes knowledge engineers are tempted to use *native* functions. For instances, the following sentence in Prolog: `min(X,Y,Min)` is not an (pure) knowledge item, as matter of fact, this is a sentence that should be evaluated in order to obtain the `Min` value. It means the verification method needs to know not only the programming language syntax but also the meaning (semantic) in order to evaluate the functions. This step is particularly important for any variables that are updated during the inference process.

IV. VERITAS, one tool for automatic verification

In what concerns SPARSE, there were clearly two main reasons to start its verification. First, the SPARSE development team carried out a set of tests based on previously collected real cases and some simulated ones . Despite the importance of these tests for the final product acceptance, the major criticism that could be pointed to this technique is that it only assures the correct performance of SPARSE under the tested scenarios.

Moreover, the tests performed during the Validation phase, namely the field tests, were very expensive since they required the assignment of substantial technical personnel and physical resources for their execution (e.g. transmission lines and coordination staff). Obviously, it is unacceptable to perform those tests for each KB update. Under these circumstances, an automatic verification tool could offer an easy and inexpensive way to assure knowledge quality maintenance, assuring the consistency and completeness of represented knowledge.

A specific tool, named VERITAS has been developed to be used in the SPARSE verification. It performs structural analysis in order to detect knowledge anomalies. VERITAS is basically composed by a module to convert the rule base, a module to compute the rule expansions and detect anomalies and another module to produce readable results through the usage of hypergraphs, notice that user is can interact with all stages of verification process (see Fig.2).

Originally, VERITAS used a non temporal knowledge base verification approach, although it proved to be very efficient in other KBS verification, regarding SPARSE, some important limitations were detected.

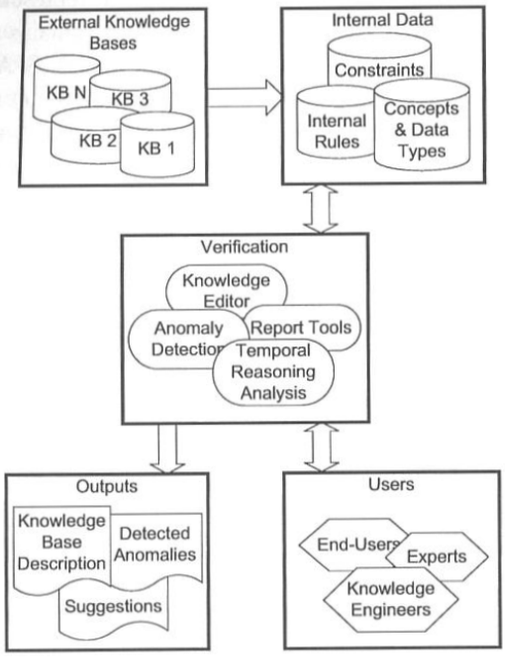

Fig. 2. VERITAS Architecture

VERITAS's main goal is detect and report anomalies, allowing the users decide whether reported anomalies reflect knowledge problems or not. Basically, anomaly detection consists in the computation of all possible inference chain that could be produced during KBS performance. Later, the inference chains (expansions) are tracked in order to find out if some constraint were violated. Notice that an expansion is an inference chain that could be generated during system functioning. For verification purposes, an *expansion* could be explicitly computed and stored.

After a filtering process, which includes *temporal reasoning analysis* and *variable evaluation* the system decides when to report an anomaly and eventually suggest some repair procedure. VERITAS is knowledge domain and rule grammar independent, due to these properties (at least theoretically), VERITAS could be used to verify any rule-based systems.

A. Data Extraction and Classification

An internal database is built for verification purposes based on the information extracted from rules. VERITAS converter uses one specific filter for each KB to be verified, this filter includes the rules for conversion between external and internal KB.

Some of the tasks performed during this step are:

- Generation of a new rule for each conjunction on LHS;
- Replacing the variables used in rules and triggers with VERITAS symbols;
- Representation of rules in a relational form in order to create indexes to speed up the expansion rules generation;

Consider the following example were one simplified SPARSE rule *d1* is converted:

```
RULE d1:
'three-phase tripping':
[[
  msg([I1,P1,[I2,NL]],'>>>tripping','01') AT T1
  AND
  breaker(I1,P1,_,unidentified,open) AT T2
  AND
  diff_less_or_equal(T2,T1,30)
  AND
  msg([I1,P1,[I2,NL, _ ]],'breaker','00') AT T3
  AND
  abs_diff_less_or_equal(T1,T3,30)
 ] OR
 [...]
]
==>
[...
 assert(breaker(I1,P1,_,tripping,open) AT T2),
 retract(msg([I1,P1,[I2,NL]],'tripping','01') AT T1),
 ...
].
```

During the extraction and classification stage the following tuples are created.

The tuple *rule/3* stores the relation between the original rule and all the new converted rules, notice, each conjunction in LHS generates a new rule.

rule(OldRule,NewRule,Description)

OldRule The identifier for the original rule;
NewRule The identifier for the new rule based on the *OldRule* plus the identification for the LHS conjunction;
Description The original rule description;
consider the following example:

```
rule(d1,'d1-11','three-phase tripping')
rule(d1,'d1-12','three-phase tripping')
```

The tuple *literal/3* stores the relation between the internal label that each literal and the external value.

literal(Label,Type,Functor/NArgs)
Label The identifier for the condition present on LHS;
Type One *literal* type can have one the following values: *time*, *status*, *event* or *userInterface*, meaning respectively that it is a, temporal operator, literal about the status of an entity,literal about the an occurrence that changes the status of an entity, literal about some message used in the user interface.
Functor/NArgs The name of *literal* and number of arguments;
consider the following example:

```
literal(tr1,time,abs_diff_less_or_equal/3).
literal(st5,status,disj_aberto_disparo/2).
...
```

The tuple *lhs/6* stores the relation between the LHS and the literals that it contains, notice that there is a tuple entry for each literal.

lhs(Rule,Label,Pos,Args,TimeTag,LogValue)

Rule Rule identifier;
Label Literal (Condition) identifier, same as the *Label* present on the tuple *literal*;
Pos Gives the position where the condition appears on LHS;
Args List of arguments used in the condition;
TimeTag Identifier for the temporal argument, just for the time tagged conditions;
LogValue Logical value of the condition, it can be *true* or *false*;
Consider the following example:

```
lhs(d1,ev2,1,
[[#I1,#P1,[#Inst2,#NL]],'tripping','01'],
#T1,true).
```

The tuple *rhs/6* is quite similar to *lhs/6* but it keeps the relation between the RHS and the literals that it contains. Besides that, there is no *LogValue* since every conclusion is true.

rhs(Rule,Label,Pos,Args,TimeTag,Group)

Label Conclusion identifier, same as the *Label* present on the tuple *literal*;
Group Used to describe different kind of conclusions, *Group* can have one the following values: *cf*, *rf* or *ui*, meaning respectively, asserting conclusions, retracting facts and user interface;
consider the following example:

```
rhs(d1,st5,7,['#I1#','#P1#'],'#T1#',true,cf).
```

The tuple *localMatch/3* contains the relation between every pair of literals that could be matched during the inference process.

localMatch(Label,(Rule1,Pos1),(Rule2,Pos2))

Label The label for the literal that matches;
(Rule1,Pos1) The first pair of rule and position where *Label* appears;
(Rule2,Pos2) The second pair of rule and position where *Label* appears;
consider the following example (see Figure 3):

```
localMatch(st11,('e1-L1',1),(d2,1)).
```

VERITAS considers some types of constraints (also referred as *impermissible sets*) during anomaly detection, some these already described in literature [22]. A constraints can be one following types:
Semantic Constraints, this type of impermissible set is formed by literals that cannot be present at the same time in the KB. Semantic constraints have to be introduced by the user. The following example concerns an electrical substation operation mode:

```
{remote(Time,Installation),
 local(Time,Installation)}
```

Single Value Constraints (SVC), this type of impermissible set is formed by only one literal but considering different values of its parameters. Those potential constraints are automatically detected and then can be either accepted, refused or modified by users. The following example shows an automatically detected SVC:

```
{
 breaker(Time,Installation,Device,open),
 breaker(Time,Installation,Device,closed),
 breaker(Time,Installation,Device,changing)
}
```

Logical Constraints, there are just two types of logical constraints (where A stands for a literal):

```
{A and not(A)}
{A and notPhysical(A)}
```

the designation $notPhysical$ is obtained by analogy with logical negation and allows to represent the constraint defined by a literal and its retraction from the KB.

After generate all the previously described data, VERITAS can show the rule dependencies through a directed hypergraph type representation (see Fig.3). This technique allows rule representation in a manner that clearly identifies complex dependencies across compound clauses in the rule base and there is a unique directed hypergraph representation for each set of rules [14].

In the following example, a set of rules ('e1-L1',d1,d2,d10) is considered.

B. Anomaly Detection

As it has been described before SPARSE has some specific features. Regarding these features the used technique is a variation of common Assumption based Truth Maintenance System (ATMS) [6]. Namely, the knowledge represented in the meta-rules had to be considered in rule expansions generation.

In the previous work of the authors [16], during rule expansions calculation, one data structure was created for each possible inference chain. Currently, the $localMatch/3$ keeps information about all predecessors of every literal, more precisely, the list of all rules that could infer this literal. It means that

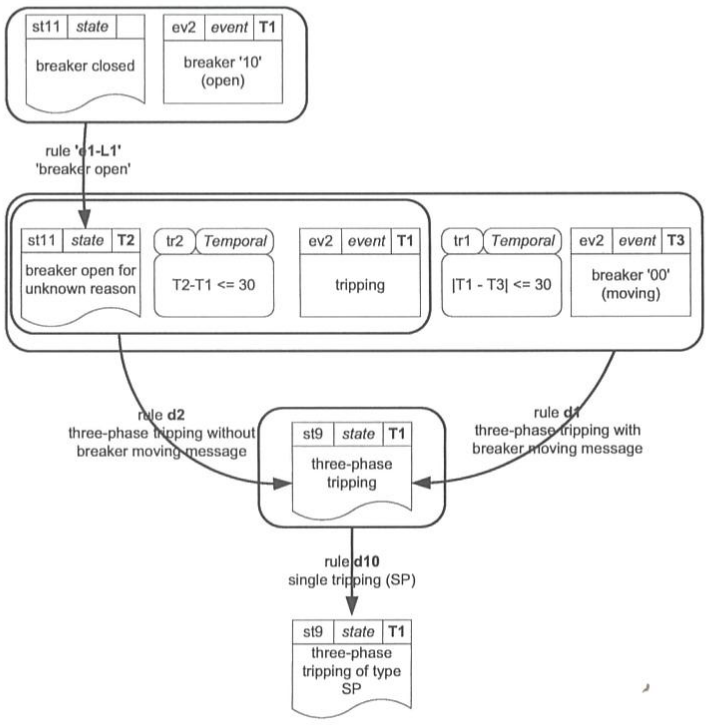

Fig. 3. Hypergraph representation

it is possible with a graph-visiting algorithm to compute the desired expansions. This approach showed to be more faster (since some tests do not need to know all rule expansions) and data flexible (e.g. circular chains became possible to exist in the verification database).

The VERITAS verification tool is able to detect anomalies existing in KB in a first step using logic tests. The detected *logic* anomalies could be grouped in three groups: redundancy, circularity and ambivalence Later, the verification algorithm performs temporal reasoning analysis which allows to filter previously detected anomalies, providing more accurate information, namely by preventing false anomaly reporting.

For instances, in the previous example (see Figure:3), two expansions would be generated:

```
(st11+ev2) ->
'e1-L1':(st11+tr2+ev2+tr1+ev2)->
d1:(st9)->  d10:(st9)

(st11+ev2) ->  'e1-L1':(st11+tr2+ev2)->
d2:(st9)->  d10:(st9)
```

Notice that both inference chains allow to conclude the literal $st9$, but the conditions set used in the first chain includes the second set, so, if only logical analysis is performed one *false* anomaly would be reported, namely, a particular type of redundancy, named as *Subsumed or Duplicate Rule*, but if the following triggers are considered, this report would be avoided.

```
trigger(ev2,[breaker open],
   [(d1,30,50),(d2,31,51),(d3,52,52)] )
```

Thus, the meta rule selects the specialist rule $d1$ first and later the general rule $d2$.

C. Anomalies Classification

Regarding that the anomalies definition depends on KBS characteristics, let us consider some definitions for the most important types of anomalies:

Definition IV.1 (Circularity) *A knowledge base contains circularity if and only if it contains a set of rules, which allows an infinite loop during rule triggering.*

Definition IV.2 (Redundancy) *A knowledge base is redundant if and only if the set of final hypotheses is the same in the rule/literal presence or absence.*

Definition IV.3 (Ambivalence) *A knowledge base is ambivalent if and only if, for a permissible set of conditions, it is possible to infer an impermissible set of hypotheses.*

Definition IV.4 (Deficiency) *A knowledge base is deficient if and only if there is not a rule that uses a set of conditions representing a possible domain knowledge situation.*

The considered is based on Preece's classification [13] with some modifications.

Namely, the matching values are considered in rule analysis, meaning that a new set of anomalies will arise. Let us consider the following circular rules:

```
RULE r1:[t(a) AND r(X)] ==> [s(a)].
RULE r2:[s(a)] ==> [r(a)].
```

For $X = a$ some inference engines could start an infinite loop.

Another situation concerns to redundancy between groups of rules. Consider the following example:

```
RULE r1:[a AND b AND c] ==> [z].
RULE r2:[NOT a AND c] ==> [z].
RULE r3:[NOT b AND c] ==> [z].
```

the rules $r1$, $r2$ and $r3$ are equivalent to the following logical expression:

$$(a \wedge b \wedge c) \vee (\neg a \wedge c) \vee (\neg b \wedge c) \rightarrow z \qquad (1)$$

Applying logical simplifications to ((1)), it is possible to obtain the following rule:

```
RULE rx: [c] ==> [z].
```

These situations can be detect using an algorithm for logical expressions simplification. Basically, this algorithm works as follows:

1. for the conclusion X, compute the set of LHSs of all rules that allows to infer X;

2. try simplify disjunction of the set obtained in the previous step;
3. compare the original expression with the simplified one, if the former is simpler (less variables or less terms) then report a redundancy anomaly;

This algorithm also works with multi-valued logic, for instances, consider the following example, notice that there are just three valid values for the b parameter's: *open*, *closed* and *changing*.

```
RULE r5:[a AND b(open)] ==> [z].
RULE r6:[a AND b(closed)] ==> [z].
RULE r7:[a AND b(changing)] ==> [z].
```

the rules $r5$, $r6$ and $r7$ are equivalent to the following logical expression:

$$a \wedge (b(changing) \vee b(closed) \vee b(open)) \rightarrow z \quad (2)$$

Applying logical simplifications to ((2)), it is possible to obtain the following rule:

```
RULE ry: [a] ==> [z].
```

Notice that redundancy between groups of rules is a generalization of the unused literal situation already studied by Alun de Preece.

D. Variables Evaluation

Efficient variables evaluation is a crucial aspect during expansion computation in order to obtain acceptable results during anomaly detection. The used technique relies on variable name replacing in the original rule by a string that uniquely represents the argument (variable). Later, during rule expansion generation this information will be used to check possible matches.

Different match types could be used during evaluation (see Fig.4), notice that *Optimal* match would correspond to the one that is obtained during system functioning:

Open In this type of match, free variables could be instantiated with terminals, consider the following example:

```
{disconnector(X), disconnector(open)}
```

where X could take *open* value. In this type of match, a large number of expansions is considered and consequently a considerable number of *false* anomalies could be detected;

Close In this case it is not allowed that free variables would match with terminals (in opposition to the *open* previous example). The reduced number of rule expansions generated will make the detection process simpler but the trade-off is that some anomalies could remain undetected;

VERITAS The used approach intends to obtain the best of the two previous kinds of options, being *Open* in order to allow detection of all possible anomalies but at same time generate the smallest possible number of expansions. How is that achieved? First, a free variable could just be replaced by valid values in the knowledge domain, previously defined in the database (e.g. $breaker(X)$, in the SPARSE domain could have the following values: *open*, *close*, *changing*). Secondly, the rules expansion are computed as late as possible and kept in a graph structure, meaning that, in the graph the following pairs would be kept (where $a \longleftrightarrow b$ means that a matches with b):

$$breaker(X) \longleftrightarrow breaker(open) \quad (3)$$

$$breaker(close) \longleftrightarrow breaker(X) \quad (4)$$

But later, during expansion rules computation, a similar process to Prolog instantiation mechanism is used, so that each variable instantiation is reflected in the following transitions. This means that an expansion that uses both relations ((4)) and ((3)) would not be generated, since that, after X becomes instantiated with *open* it could not be later instantiated with *close*.

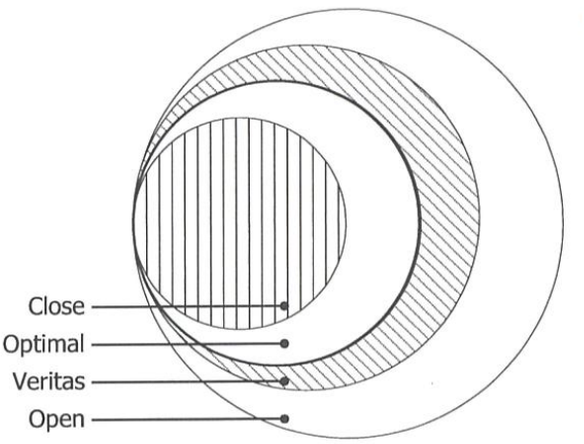

Fig. 4. Match Schema

V. Conclusions and Future Work

This paper focussed on some aspects of the practical use of KBS in Control Centers, namely, knowledge maintenance and its relation to the Verification process. It becomes clear to the development team that the use of Verification tools (based on formal methods), increases the confidence of the end users and eases the process of maintaining a knowledge base. It also reduces the testing costs and the time needed to implement those tests.

This paper described VERITAS a verification tool that performs the structural analysis in order to detect knowledge anomalies. During its tests the SPARSE KBS was used. Some characteristics that mostly constrained the development and use of verification tool were pointed like, temporal reasoning and variables evaluation, hence, the adopt solutions were described.

VI. References

[1] B.W. Boehm. Verifying and validating software requirements and design specifications. *IEEE Software*, 1(1):75–88, 1984.

[2] B. Cragun and H. Steudel. A decision table based processor for checking completeness and consistency in rule based expert systems. *International Journal of Man Machine Studies (UK)*, 26(5):633–648, May 1987.

[3] A. Gerevini. Reasoning about time and actions in artificial intelligence: Major issues. In O.Stock, editor, *Spatial and Temporal Reasoning*, pages 43–70. Kluwer Academic Pblishers, 1997.

[4] A. Ginsberg. A new aproach to checking knowledge bases for inconsistency and redundancy. *In procedings 3rd Annual Expert Systems in Government Conference*, pages 102–111, October 1987.

[5] A. Gonzalez and D. Dankel. *The Engineering of Knowledge Based Systems - Theory and Practice*. Prentice Hall International Editions, 1993.

[6] J. Kleer. An assumption-based TMS. *Artificial Intelligence Holland*, 2(28):127–162, 1986.

[7] T. Menzies. Knowledge maintenance: The state of the art, the knowledge. *Engineering Review*, 1998.

[8] D.L. Nazareth. Investigating the applicability of petri nets for rule based systems verification. *IEEE Transactions on Knowledge and Data Engineering*, 4(3):402–415, 1993.

[9] T. Nguyen, W. Perkins, T. Laffey, and D. Pecora. Knowledge Based Verification. *AI Magazine*, 2(8):69–75, Summer 1987.

[10] E. Pipard. Detecting inconsistencies and incompleteness in rule bases: the INDE system. In *Proceedigns of the 8th International Workshop on Expert Systems and their Applications, 1988, Avignon, France;*, volume 3, pages 15–33, Nanterre, France, 1989. EC2.

[11] A. Preece. Building the right system right. In *Proc.KAW'98 Eleventh Workshop on Knowledge Acquisition, Modeling and Management*, 1998.

[12] A. Preece, R. Bell, and C. Suen. Verifying knowledge-based systems using the cover tool. *Proccedings of 12th IFIP Congress*, pages 231–237, 1992.

[13] A. Preece and R. Shinghal. Foundation and application of knowledge base verification. *International Journal of Intelligent Systems*, 9(8):683–702, 1994.

[14] M. Ramaswamy and S. Sarkar. Global verification of knowledge based systems via local verification of partitions. In *Proceedings of the European Symposium on the Verification and Validation of Knowledge Based Systems*, pages 145–154, Leuven,Belgium, 1997.

[15] M. Rousset. On the consistency of knowledge bases:the COVADIS system. In *Procedings of the European Conference on Artificial Intelligence (ECAI'88)*, pages 79–84, Munchen, August 1988.

[16] J. Santos, L. Faria, C. Ramos, Z. Vale, and A. Marques. Veritas - a verification tool for real-time applications in power system control centers. In *Proc.12th International Florida AI Research Society (FLAIRS'99)*, pages 511–515, Orlando, USA, 1999. AAAI Press.

[17] M. Suwa, A. Scott, and E. Shortliffe. An aproach to verifying completeness and consistency in a rule based expert system. *AI Magazine (EUA)*, 3(4):16–21, 1982.

[18] Z. Vale, A. Moura, M. Fernandes, and A. Marques. Sparse - an expert system for alarm processing and operator assistance in substations control centers. *ACM (Association for Computing Machinery) Press - Applied Computing Review*, 2(2):18–26, December 1994.

[19] J. Vanthienen, C. Mues, and G. Wets. Inter Tabular Verification in an Interactive Environment. *Proceedings of the European Symposium on the Verification and Validation of Knowledge Based Systems*, pages 155–165, 1997.

[20] L. Vila. A survey on temporal reasoning in artificial intelligence. *AI Communications*, 7(1):4–28, 1994.

[21] Chih-Hung Wu and Shie-Jue Lee. Enhanced High-Level Petri Nets with Multiple Colors for Knowledge Verification/Validation of Rule-Based Expert Systems. *IEEE TRansactions on Systems. Man, and Cybernetics*, 27(5):760–773, October 1997.

[22] N. Zlatareva and A. Preece. An effective logical framework for knowledge based systems verification. *International Journal of Expert Systems*, 7(3):239–260, 1994.