24

The Verification of Temporal KBS: SPARSE - A Case Study in Power Systems

Jorge Santos[1], Zita Vale[2], Carlos Serôdio[3] and Carlos Ramos[1]
[1]*Departamento de Engenharia Informática, Instituto Superior de Engenharia do Porto,*
[2]*Departamento de Engenharia Electrotécnica, Instituto Superior de Engenharia do Porto,*
[3]*Departamento de Engenharias, Universidade de Trás-os-Montes e Alto Douro,*
Portugal

1. Introduction

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 a hard and complex task. Furthermore, albeit the last significant developments in temporal reasoning and representation (TRR) area, there still is a considerable gap for its successful use in practical applications.

In this chapter we present VERITAS, a tool that focus time maintenance, that is one of the most important processes in the engineering of the time during the development of KBS.

The verification and validation (V&V) process is part of a wider process denominated knowledge maintenance (Menzies 1998), in which an enterprise systematically gathers, organizes, shares, and analyzes knowledge to accomplish its goals and mission. The V&V process states if the software requirements specifications have been correctly and completely fulfilled.

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

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.

The rest of the chapter is structured as follows. Section 2 provides a short overview of the state-of-art of V&V and its most important concepts and techniques. After that, section 3 describes SPARSE, a KBS used to assist the Portuguese Transmission Control Centres operators in incident analysis and power restoration. Special attention is given to SPARSE's particular characteristics, introducing the problem of verifying real world applications. Section 4 presents VERITAS; special emphasis is given to the tool architecture and to the method used in anomaly detection. Finally, in section 5, achieved results are discussed and in section 6, we present some conclusions and ideas for future work.

Source: Robotics, Automation and Control, Book edited by: Pavla Pecherková, Miroslav Flídr and Jindřich Duník,
ISBN 978-953-7619-18-3, pp. 494, October 2008, I-Tech, Vienna, Austria

2. Verification and validation of KBS

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 Knowledge Based Systems (KBS) should undergo the same evaluation process. Besides, 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. Unfortunately, the methodologies proposed in software engineering have showed to be inadequate for knowledge based systems validation and verification, since KBS present some particular characteristics (Gonzalez & Dankel 1993). Namely, the need for KBS to deal with uncertainty and incompleteness; the domains modelled normally do not underline physical models; it is not rare for KBS to have the ability to learn and improve the KB allowing a dynamical behaviour; in most domains of expertise, there is no concept of right results but only of acceptable ones.

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.

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.

Albeit there is no general agreement on the V&V terminology (Hoppe & Meseguer 1991), the following definitions will be used in the scope of this paper.

- **Validation** - Validation means building the right system (Boehm 1984). 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;
- **Verification** - Verification means building the system right (Boehm 1984). 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.
- **Verification & Validation** - The Verification and Validation process allows determining if the requirements have been correctly and completely fulfilled in order to assure the system's reliability, safety, quality and efficiency. More synthetically, it can be said that the V&V process is to build the right system right (Preece 1998).

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 (Preece 1998). The efficiency of these techniques strongly depends on the existence of test cases or on the degree of formalization used in the specifications. One of the most used techniques is static verification, which consists of sets of logical tests executed in order to detect possible knowledge anomalies.

- **Anomaly –** An anomaly is a symptom of one (or multiple) possible error(s). Notice that an anomaly does not necessarily denote an error (Preece & Shinghal 1994).

Rule 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. The rule base should reflect the nature of this process, meaning that if documentary sources are used, the rule base should reflect knowledge sources. Consequently, some anomalies are desirable and intentionally inserted in KB. For instance, redundancy on the documentary sources will lead to redundant KB. Rule based systems are still the more often used representation in the development of KBS. The scientific community has deeply studied 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. Some well known V&V tools used different techniques to detect anomalies. The KB-Reducer (Ginsberg 1987) system represents rules in logical form, and 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 become grounded. The COVER (Preece, Bell & Suen 1992) works in a similar fashion using the ATMS (Assumption Truth Maintaining System) approach (Kleer 1986) and graph theory, allowing the detection of a large number of anomalies. The COVADIS (Rousset 1988) successfully explored the relation between input and output sets. The ESC (Cragun & Steudel 1987), RCP (Suwa, Scott & Shortliffe 1982) and Check (Nguyen et al. 1987) systems and more recently the PROLOGA (Vanthienen, Mues & Wets 1997) used decision table methods for verification purposes. This approach proved to be quite interesting, especially when the systems to be verified also used decision tables as representation support. These systems' major advantage is that it enables tracing the reasoning path quite clearly, while the major problem is the lack of solutions for verifying long reasoning inference chains. Some authors studied the applicability of Petri nets (Pipard 1989; Nazareth 1993) to represent the rule base and to detect the knowledge inconsistencies. More recently coloured Petri nets were used (Wu & Lee 1997). Although specific knowledge representations provide higher efficiency while used to perform some verification tests, arguably all of them could be successfully converted into production rules.

3. The case study: SPARSE

Control Centres (CC) are very important in the operation of electrical networks when receiving real-time information about network status. CC operators should take, usually in a short time, the most appropriate actions in order to reach the maximum network performance.

In case of incident conditions, a huge volume of information may arrive to these centres. The correct and efficient interpretation by a human operator becomes almost impossible. In order to solve this problem, some years ago, electrical utilities began to install intelligent applications in their control centres (Amelink, Forte & Guberman 1986; Kirschen & Wollenberg 1992). These applications are usually KBS and are mainly intended to provide operators with assistance, especially under critical situations.

3.1 Architecture and functioning

SPARSE (Vale et al. 1997) is a KBS used in the Portuguese Transmission Network (REN) for incident analysis and power restoration. In the beginning it started to be an expert system

(ES) and it was developed for the Portuguese Transmission Network (REN) Control Centres. The main goals of this ES were to assist Control Centre operators in incident analysis, allowing a faster power restoration. Later, the system evolved to a more complex architecture (Vale et al. 2002), which is normally referred as a Knowledge Based System (see Fig. 1).

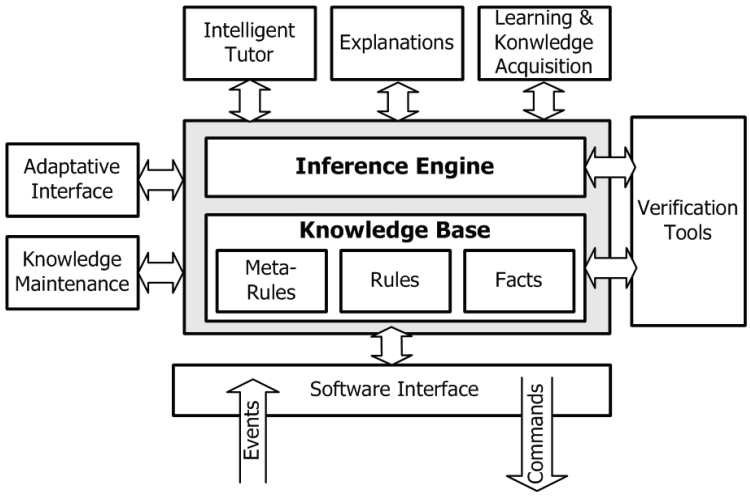

Fig. 1 - SPARSE Architecture

SPARSE includes many modules, namely for learning and automatic data acquisition (Duarte et al. 2001), adaptive tutoring (Faria et al. 2002) and automatic explanations (Malheiro et al. 1999). As it happens in the majority of KBSs, one of the most important SPARSE components is the knowledge base (KB) (see formula (1)):

$$KB = RB \cup FB \cup MRB \tag{1}$$

where:

- RB stands for rule base;
- FB stands for facts base;
- MRB stands for meta-rules base;

The rule base is a set of clauses with the following structure:

```
RULE ID: 'Description':
 [
  [C1 AND C2 AND C3]
  OR
  [C4 AND C5]
 ]
 ==>
 [A1,A2,A3].
```

The rule's Left Hand Side (LHS) is a set of conditions (C1 to C5 in this example) of the following types:

- A fact, representing domain events or status messages. Typically these facts are time-tagged;

- A temporal condition over facts;
- Previously asserted conclusions.

The rule's Right Hand Side (RHS) is a set of actions/conclusions to be taken (A1 to A3 in this example) and may be of one of the following types:

- Assertion of facts representing conclusions to be inserted in the knowledge base. A conclusion can be final (e.g., a diagnosis) or intermediate (e.g., a status fact concerning that later it will be used in other rule LHS);
- Retraction of facts (conclusions to be deleted from the knowledge base);
- Interaction with the user interface.

Let's consider the rule d3 as an example of a SPARSE rule:

```
rule d3 : 'Monophasic Triggering':
 [
  [
   msg(Dt1,Tm1,[In1,Pn1,[In2,NL]],'>>>TRIGGERING','01') at T1 and
   breaker(_,_,In1,Pn1,_,_, closed) and
   msg(Dt2,Tm2,[In1,Pn1,[In2,NL, _ ]],'BREAKER','00') at T2 and
   condition(abs_diff_less_or_equal(T2,T1,30))
  ]
  or
  [
   msg(Dt1,Tm1,[In1,Pn1,[In2,NL]],'>>>TRIGGERING','01') at T1 and
   msg(Dt2,Tm2,[In1,Pn1,[In2,NL]],'BREAKER','00') at T2  and
   condition(abs_diff_less_or_equal(T2,T1,30))
  ]
 ]
 ==>
 [
  assert(triggering(Dt1,Tm1,In1,Pn1,In2,NL,monophasic,not_identified,T2),T1),
  retract(breaker(_,_,In1,Pn1,_,_,closed),_,T2),
  assert(breaker(Dt2,Tm2,In1,Pn1,_,triggering,mov),T2),
  retract(msg(Dt1,Tm1,[In1,Pn1,[In2,NL]],'>>>TRIGGERING','01'),T1,T1),
  retract(msg(Dt2,Tm2,[In1,Pn1,[In2,NL | _ ]],'BREAKER','00'),T2,T2),
  retract(breaker_opened(In1,Pn1),_,T1),
  assert(breaker_opened(In1,Pn1),T1)
 ].
```

The meta-rule base is a set of triggers, used by the rule selection mechanism, with the following structure:

$$\text{trigger}(Fact, [(R_1, TB_1, TE_1), \ldots, (R_n, TB_n, TE_n)]) \quad (2)$$

standing for:

- Fact - the arriving fact (external alarm or a previously inferred conclusion);
- (R_i, TB_i, TE_i) - the temporal window were the rule R_i could by triggered. TB_i is the delay time before rule triggering, used to wait for remaining facts needed to define an event, and the TE_i is the maximum time for trying to trigger the rule R_i.

The inference process relies on the cycle depicted in Fig. 2. In the first step, SPARSE collects a message (represented as a fact in SPARSE scope) from SCADA[1], then the respective trigger is selected and some rules are scheduled. The scheduler selects the next rule to be tested (the inference engines try to prove its veracity). Notice that, when a rule succeeds, the conclusions (on the RHS) will be asserted and later processed in the same way as the SCADA messages.

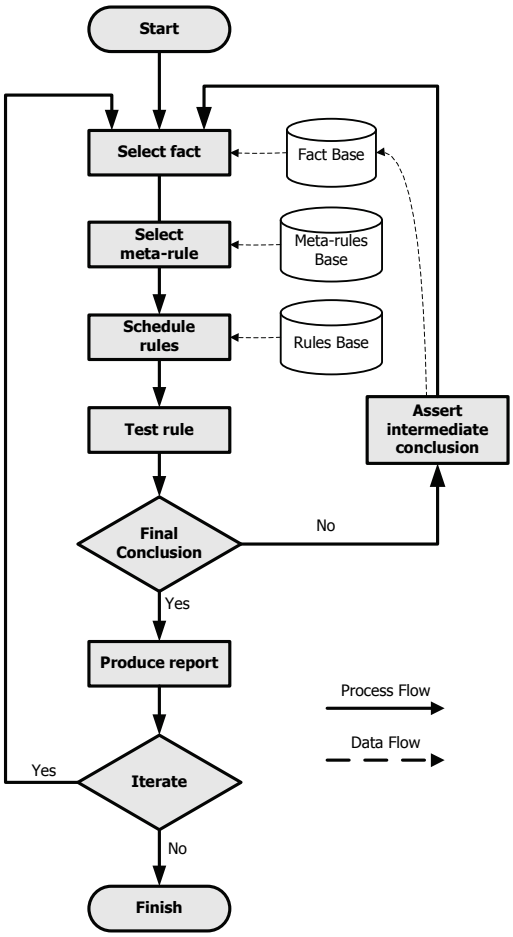

Fig. 2 - SPARSE main algorithm

Let's consider the following meta-rule that allows scheduling the rule d3:

trigger(msg(_,_,[Inst1,Painel1,_],'>>>TRIGGERING','01'), [(d1,30,50),(d2,31,51),(d3,52,52)]).

[1] Supervisory Control And Data Acquisition: this system collects messages from the mechanical/electrical devices installed in the network.

The use of rule selection mechanism allows configuring a heuristic approach with the following characteristics:

- **Problem space reduction –** since it assures that only a set o rules related to the Fact will be used and tested;
- **From specific to general rules –** the temporal windows allow defining an explicit order, so the system usually proceeds from the most specific to general.

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 out 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 would be unacceptable to perform those tests after each KB update. Under these circumstances, an automatic verification tool could offer an easy and inexpensive way of assuring knowledge quality maintenance, assuring the consistency and completeness of represented knowledge.

3.2 The verification problem

The verification problem based on the anomaly detection usually relies on the calculation of all possible inference chains 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 that make the verification work harder. 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. The following ones are the most important:

- **Rule triggering selection mechanism** - In what concerns SPARSE, this mechanism was implemented using both meta-rules and the inference engine. As for 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;
- **Temporal reasoning** - This issue received large attention from the scientific community in last two decades (surveys covering this issue can be found in (Gerevini 1997; Fisher, Gabbay & Vila 2005)). Although *time* is ubiquitous in society, and despite 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 major challenge is to reason about events. Therefore, it is necessary to deal with time intervals (e.g., temporal windows of validity), points (e.g., instantaneous events occurrence), alarms order, duration and the presence or/and absence of data (e.g., messages lost in the collection or/and transmission system);

- **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;
- **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 encode rule using procedural predicates. For instance, the following sentence in Prolog: `min(X,Y,Min)` calls a procedure that compares `X` and `Y` and instantiates `Min` with smaller value. Thus, it is not a (pure) knowledge item; in terms of verification it should be evaluated in order to obtain the `Min` value. It means that the verification method needs to consider 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.

4. The verification tool: VERITAS

VERITAS is an automatic tool developed for KBS verification. This tool performs KB structural analysis allowing knowledge anomalies detection. Originally, VERITAS used a non temporal KB verification approach. Although it proved to be very efficient in other KBS verification – in an expert system for cardiac diseases diagnosis (Rocha 1990) and in other expert system otology diseases diagnosis and therapy (Sampaio 1996) –, in SPARSE case some important limitations were detected.

4.1 Main process

The VERITAS main process, depicted in Fig. 3, relies on a set of modules that assures the following competences: converting the original knowledge base; creating the internal knowledge base; computing the rule expansions and detecting anomalies; producing readable results. Regarding that, the user can interact with all stages of the verification process.

The conversion module translates the original knowledge base files into a set of new ones containing the same data but represented in an independent format, recognized by VERITAS. For this module functioning, a set of conversion rules is also needed, specifying the translation procedure. This module assures the independence of VERITAS to the format and syntax used in specification of the KB to be verified. The conversion operations largely depend on the original format, although the most common conversion operations are:

- If the LHS is a disjunctive form, a distinct rule is created for each conjunction contained by the LHS. The rule RHS remains the same;
- Original symbols, like logical operators, are replaced by others accordingly to the internal notation;
- The variables are replaced by internal symbols and a table is created in order to store these symbols.

During the conversion step, the rule d3 (previously presented) would be transformed in two d3-L1 and d3-L2, since the LHS contains two conjunctions. The tuple cvr/3 stores the intermediate rule d3-L1 presented after Fig.3.

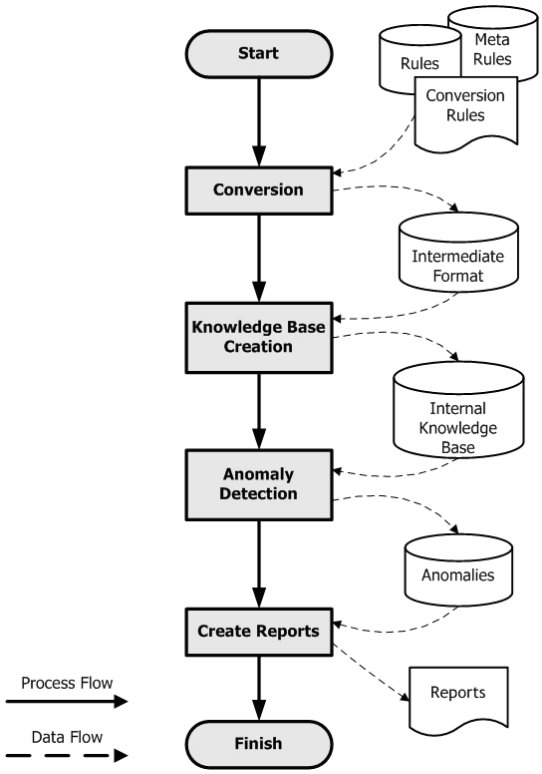

Fig. 3 - VERITAS main algorithm

```
cvr(
    d3-L1,
    Monophasic Triggering,
    [
    msg(#Dt1,#Tm1,[#In1,#Pn1,[#In2,#NL]],>>>TRIGGERING,01) at #T1,
    breaker(#_,#_,#In1,#Pn1,#_,#_, closed),
    msg(#Dt2,#Tm2,[#In1,#Pn1,[#In2,#NL,#_]],BREAKER,00) at #T2,
    abs_diff_less_or_equal(#T2,#T1,30)
    ],
    [
    assert(triggering(#Dt1,#Tm1,#In1,#Pn1,#In2,#NL,monophasic,not_identified,#T2),#T1),
    retract(breaker(#_,#_,#In1,#Pn1,#_,#_,fechado),#_,#T2),
    assert(breaker(#Dt2,#Tm2,#In1,#Pn1,#_,triggering,mov),#T2),
    retract(msg(#Dt1,#Tm1,[#In1,#Pn1,[#In2,#NL]],>>>TRIGGERING,01),#T1,#T1),
    retract(msg(#Dt2,#Tm2,[#In1,#Pn1,[#In2,#NL|#_]],BREAKER,00),#T2,#T2),
    retract(breaker_opened(#In1,#Pn1),#_,#T1),
    assert(breaker_opened(#In1,#Pn1),#T1)
 ]).
```

After the conversion step, the internal knowledge base is created (see section 4.2). This KB will store information about the following issues:

- The original rules and meta-rules, represented according to the structure that allows speeding up the expansion calculation;
- A set of restrictions over the knowledge domain modelled in KB regarding this set can be manually or semi-automatically created;
- A characterization over the data items extracted from the original KB.

In the following step, the anomaly detection (see section 4.3), the module responsible for this task will examine the KB in order to calculate every possible inference that could be entailed during KBS functioning, and then detect if any constraint (logic or semantic) is violated. Finally, the detected anomalies are reported in a suitable way for human analysis.

4.2 Knowledge base

The knowledge base schema was designed regarding the need to speed the expansion calculation since it is one of the most time consuming steps in the verification process. In Fig. 4 the concepts and relations contained in the schema are depicted.

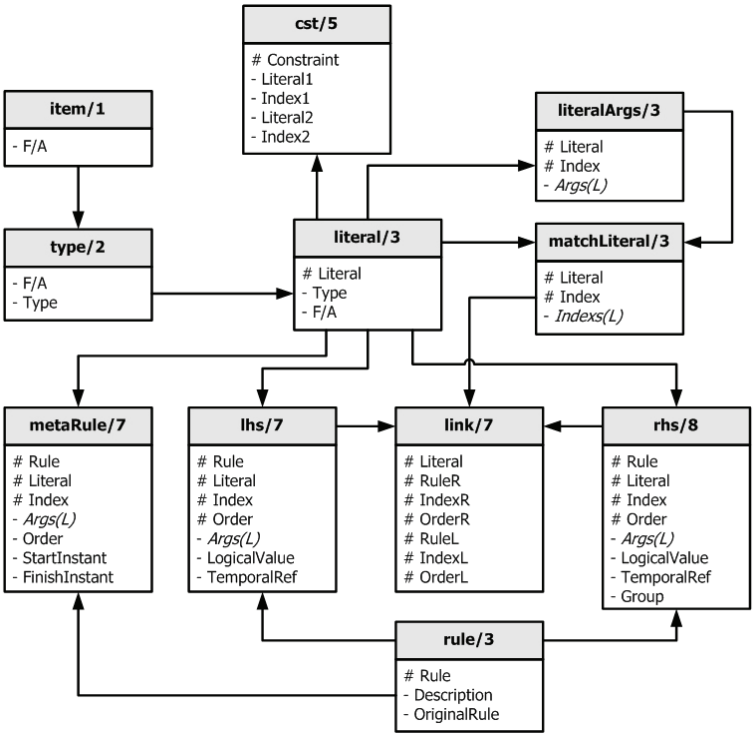

Fig. 4 - Knowledge Base Schema

This schema allows to: store the rules and meta-rules in an efficient way; classify the elements that compose such rules and meta-rules. Therefore, tuples item/1, type/2 and literal/3 allow to classify the elements (in VERITAS scope named literals) that compose both

rule LHSs and RHSs. The tuples literalArgs/3 and matchLiteral/ represent, respectively: the arguments for each literal and the pairs of literals that could be matched during expansion calculation. The tuple rule/3 relates the old and new rule representations while lhs/7 and rhs/8 store the literals that compose each rule LHS and RHS, respectively. Finally, metaRule/7 stores the information related to original meta-rules.

$$\text{item}(F \ / \ A) \tag{3}$$

The tuple item/1 stores the functor and arity of each literal appearing in rules LHS and RHS contained in RB. For the rule d3-L1, the following items would be asserted:

item(abs_diff_less_or_equal/3).
item(breaker_opened/2).
item(triggering/9).
item(breaker/7).
item(msg/5).

$$\text{type}(F \ / \ A, Type) \tag{4}$$

The tuple type/2 allows the characterization of the items previously extracted and stored in item/1. The classification can be done manually or semi-automatically. The field Type can exhibit one of the following values:

- **interface –** an operation for user interface;
- **status –** state of a knowledge domain element (e.g., electrical devices);
- **process –** used to define eventualities with duration (non-instantaneous);
- **event** - used to define instantaneous eventualities;
- **time –** temporal operator used to reason about time;
- **operation** - used to define "procedural" operations like comparisons.

The following tuples type/2 would be created for the considered rule d3-L1:

type(abs_diff_less_or_equal/3,time).
type(breaker_opened/2,status).
type(triggering/9,status).
type(breaker/7,status).
type(msg/5,event).

$$\text{literal}(Literal, Type, F \ / \ A) \tag{5}$$

The tuple literal/3 stores the synthesis of type/2 and item/1. Besides, it allows labelling the literals with a key (Literal) for what they will be referred during the verification process. According to the example, the following instances of literal/3 would be created:

literal(tr1,time,abs_diff_less_or_equal/3).
literal(st5,status,breaker_opened/2).
literal(st9,status,triggering/9).
literal(st11,status,breaker/7).
literal(ev1,event,msg/5).

$$\text{literalArgs}(Literal, Index, Args) \tag{6}$$

During the expansions calculation, VERITAS replaces each literal X contained in a LHS rule by the literals that compose the LHS of the rule that allows inferring a literal Y if the literal Y matches X. Actually, this process simulates the inference engine using backward chaining. During the KBS usage the process of matching literals is straightway since the variables values are known; however, it doesn't happen in the verification process, so the number of expansions (possible inference chains) that the system needs to calculate grows exponentially. Additionally, during the expansions calculation each pair of literals needs to be checked often, so in order to avoid it, VERITAS computes *à priori* the pairs of literals that can be matched. Henceforth, the tuple literalArgs/3 stores the diverse occurrences of *similar* literals. Two literals are *similar* if they exhibit the same functor and arity but their respective lists of arguments are not *similar*. Two lists of arguments are similar if they have the same size and for each element in corresponding position one of the following situations happens:

- The argument is a free variable, meaning it can be matched with everything;
- The argument is a terminal (not a free variable) and in this case both arguments need to exhibit the same value.

Considering the literal st9 the following instances of literalArgs/3 would be created:

```
literalArgs(st9,1,[#Dt,#Tm,#In1,#Pn1,#In2,#NL,#Type,not_identified,#T2]).
literalArgs(st9,2,[#Dt1,#Tm1,#In1,#Pn1,#In2,#NL,monophasic,not_identified,#Tabr]).
literalArgs(st9,3,[#Dt2,#Tm2,#In1,#Pn1,#In2,#NL,triphasic,dtd,#Tabr2]).
literalArgs(st9,4,[#Dt2,#Tm2,#In1,#Pn1,#In2,#NL,monophasic,dmd,#Tabr2]).
literalArgs(st9,5,[#Dt1,#Tm1,#In1,#Pn1,#In2,#NL,triphasic,rel_rap_trif,#Tabr1]).
literalArgs(st9,6,[#Dt1,#Tm1,#In1,#Pn1,#In2,#NL,triphasic,dtr,#Tabr1]).
literalArgs(st9,7,[#Dt1,#Tm1,#In1,#Pn1,#In2,#NL,monophasic,rel_rap_mono,#Tabr1]).
literalArgs(st9,8,[#Dt1,#Tm1,#In1,#Pn1,#In2,#NL,monophasic,dmr,#Tabr1]).
literalArgs(st9,9,[#Dt1,#Tm1,#In1,#Pn1,#In2,#NL,triphasic,ds,#Tabr]).
literalArgs(st9,10,[#Dt1,#Tm1,#In1,#Pn1,#In2,#NL,#_, #TriggeringType,#Tabr]).
literalArgs(st9,11,[#Dt2,#Tm2,#In1,#Pn1,#In2,#NL,triphasic,not_identified,#Tabr]).
literalArgs(st9,12,[#Dt2,#Tm2,#In1,#Pn1,#In2,#NL,triphasic,close_defect,#Tabr]).
```

$$\text{matchLiteral}(Literal, Index, Indexes) \tag{7}$$

The tuple matchLiteral/3 stores the possible matches between a literal defined by pair Literal/Index and a list of indexes. Notice that the process used to determine the possible matches between literals is similar to determining a Graph Transitive Closure where the pair Literal/Index is each graph node and the list Indexes is the set of adjacent edges. In the considered example the following instances of matchLiteral/3 would be created:

```
matchLiteral(st9,1,[1,2,10,11]).
matchLiteral(st9,2,[1,2,10]).
matchLiteral(st9,3,[3,10]).
matchLiteral(st9,4,[4,10]).
matchLiteral(st9,5,[5,10]).
matchLiteral(st9,6,[6,10]).
matchLiteral(st9,7,[7,10]).
matchLiteral(st9,8,[8,10]).
matchLiteral(st9,9,[9,10]).
matchLiteral(st9,10,[1,2,3,4,5,6,7,8,9,10,11,12]).
matchLiteral(st9,11,[1,10,11]).
matchLiteral(st9,12,[10,12]).
```

$$\text{rule}(Rule, Description, OriginalRule) \quad (8)$$

The tuple rule/3 allows relating the new rules used by VERITAS and the original ones. This tuple is quite useful in the user interaction since the user is more acquainted with the original rules. For the rule d3 the following instances of rule/3 would be created:

rule(d3-L1,Monophasic Triggering,d3).
rule(d3-L2,Monophasic Triggering,d3).

$$\text{lhs}(Rule, Literal, Index, Position, TemporalArg, LogicalValue, Args) \quad (9)$$

The tuple lhs/7 stores the set of conditions forming the rule LHS. Meaning, the occurrence of a literal (Literal/Index) in a rule and its position, temporal label (TemporalArg), logical value (since a condition can be a negation of the referred literal) and arguments. Considering the rule d3-L1, the following instances of lhs/7 would be asserted:

lhs(d3-L1,ev1,8,1,#T1,true,[#Dt1,#Tm1,[#In1,#Pn1,[#In2,#NL]],>>>TRIGGERING,01]).
lhs(d3-L1,st11,8,2,none,true,[#_,#_,#In1,#Pn1,#_,#_,fechado]).
lhs(d3-L1,tr1,1,4,none,true,[#T2,#T1,30]).
lhs(d3-L1,ev1,7,3,#T2,true,[#Dt2,#Tm2,[#In1,#Pn1,[#In2,#NL,#_]],BREAKER,00]).

$$\text{rhs}(Rule, Literal, Index, Position, TemporalArg, LogicalValue, Args, Type) \quad (10)$$

The tuple rhs/8 stores the set of conclusions forming the rule RHS. This tuple has about the same structure as the lhs/7 but adds the field Type, which can exhibit the following values:

- **cf** – assertion of conclusion;
- **rf** – retraction of fact;
- **it** – user interface.

Considering the rule d3-L1, the following instances of rhs/8 would be asserted:

rhs(d3-L1,st9,1,1,#T1,true,cf,[#Dt1,#Tm1,#In1,#Pn1,#In2,#NL,monophasic,not_identified, #T2]).
rhs(d3-L1,st11,8,2,#T2,true,rf,[#_,#_,#In1,#Pn1,#_,#_,closed]).
rhs(d3-L1,st11,4,3,#T2,true,cf,[#Dt2,#Tm2,#In1,#Pn1,#_,triggering,mov]).
rhs(d3-L1,ev1,8,4,#T1,true,rf,[#Dt1,#Tm1,[#In1,#Pn1,[#In2,#NL]],>>>TRIGGERING,01]).
rhs(d3-L1,ev1,9,5,#T2,true,rf,[#Dt2,#Tm2,[#In1,#Pn1,[#In2,#NL|#_]],BREAKER,00]).
rhs(d3-L1,st5,1,6,#T1,true,rf,[#In1,#Pn1]).
rhs(d3-L1,st5,1,7,#T1,true,cf,[#In1,#Pn1]).

$$\text{metaRule}(Rule, Literal, Index, Order, StartInstant, FinishInstant, Args) \quad (11)$$

The tuple metaRule/7 stores the information the meta-rules used in the rule selection triggering mechanism. Hence, a rule is scheduled for the interval defined by StartInstant and FinishInstant if the literal defined by the pair Literal/Index is asserted in the KB. Concerning the tuple trigger/2 used by SPARSE (presented in section 3.1) and the rule d3, the following instances of metaRule/7 would be asserted:

metaRule(d3-L1,ev1,8,3,52,52,[#_,#_,[#Inst1,#Panel1,#_],>>>TRIGGERING,01]).
metaRule(d3-L2,ev1,8,4,52,52,[#_,#_,[#Inst1,#Panel1,#_],>>>TRIGGERING,01]).

$$\text{link}(RuleL, Literal, IndexL, PositionL, RuleR, IndexR, PositionR) \quad (12)$$

The tuple link/7 instantiates the tuple matchLiteral/3 for a specific knowledge base. While the matchLiteral/3 states which pairs of literals can be matched, the tuple link/7 stores which pairs of literals present in a KB can actually be matched during the expansions calculation. Concerning the literal st9 and the rule d3-L1, the following instances of link/7 would be created:

link(d3-L1,st9,2,1,d22,10,1).
link(d3-L1,st9,2,1,d21,10,2).
link(d3-L1,st9,2,1,d21,10,1).
link(d3-L1,st9,2,1,d5,2,1).
link(d3-L1,st9,2,1,i1,1,1).

$$\text{cst}(Constraint, Literal1, Index1, Literal2, Index2) \tag{13}$$

The tuple cst/5 allows storing the impermissible sets for a knowledge base. Each occurrence of this tuples states that a pair of literals is incompatible together.

4.3 Anomaly detection

The algorithm used for the anomaly detection, depicted in Fig. 5, works in the following way.

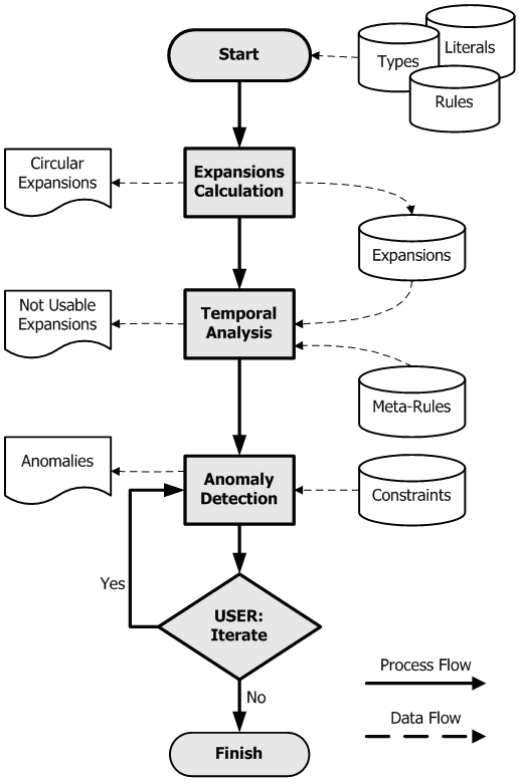

Fig. 5 - Anomaly Detection Algorithm

In the first step, the knowledge base is previously created (see section 4.2). After that, the expansions are calculated (see section 4.3.1); during this step the circular expansions are detected and labelled. In the next step, using both expansions and meta-rules, the temporal consistency of the expansions is evaluated (see section 4.3.2). Finally, specific algorithms are applied in order to detect the following anomalies: circularity (see section 4.3.3), ambivalence (see section 4.3.4) and redundancy (see section 4.3.5).

4.3.1 Expansions calculation

The expansions calculation process intends to thoroughly determine the inference chains that can be possibly drawn during the KBS functioning. Calculating an expansion consists in breadth-first search over a hypergraph, in which each hypernode is a set of literals and each transition represents a rule. The procedure calcExpansion works in the following way:

```
procedure calcExpansion(literal,expansion)
    input: literal
    output: expansion
    begin
    ;;Variables initialization
    rules = empty;
    expansion = empty;
    listE = pushBack(listE, (literal, rules));
    if ( ∃ rule(r,lhs,rhs) and (literal) ∈ rhs)) then
        (literal,rules) = popFront(listE);
        foreach(literal ∈ lhs) do
            listE = pushBack(listE,(literal,rules));
        end- foreach
        while listE ≠ empty do
            (literal,rules) = popFront(listE);
            if( ∃ rule(r,lhs,rhs) and (literal ∈ rhs)) then
                ;;literal inferable;
                rules = pushBack(rules, r);
                if(r \ in rules) then
                    ;;circular
                    expansion = pushBack(expansion, c(literal, rules));
                else
                    ;;inferable and not circular;
                    expansion = pushBack(expansion, e(literal, rules));
                    rules = pushBack(rules, r);
                    foreach literal ∈ lhs do
                        listE = pushBack(listE, (literal, rules));
                    end-foreach
                    expansion = updateLRulesExp(expansion, rules);
                end-if
            else
            ;;not inferable literal
            expansion = pushBack (expansion, f(literal));
        end-if
    end-while
end-procedure
```

First, some variables are initialized, namely: listE (the list of literals to be expanded); rules (list of rules that used in an expansion); expansion (list of the expanded literals). Hence, for each literal contained in the KB, a rule that allows its inference is selected and the literals contained in the LHS are stored in the list listE. This algorithm finishes when listE becomes empty, meaning that all literals were expanded. The elements contained in listE are iteratively popped and they can be one of the following types:

- **Not inferable –** this type of literals are ground facts or basic operations;
- **Inferable –** this type of literals can be inferred during KBS functioning. If a literal is part of a circular inference chain, the algorithm labels it and the expansion for this literal finishes; otherwise the algorithm calculates the set of literals needed to infer it and pushes this set into the listE.

Finally, the algorithm produces a list of the following kind of tuples:

- **f(literal) –** represents a literal not inferable;
- **e(literals,ruleList) –** represents a list of inferable literals and the rules used to infer them;
- **c(literals,ruleList) –** represents a list of inferable literals that configure a circular chain.

Let's consider the following set of rules:

```
rule(r1,[st1,ev1],[st2,st3])
rule(r2,[st3,ev3],[st6,ev5])
rule(r3,[ev3],[ev4])
rule(r4,[ev1,ev2],[ev4,st4])
rule(r5,[ev5,st5,ev4],[st7,st8])
```

After the use of the described algorithm over this set of rules, the following two expansions would be obtained:

```
f(ev3)
f(st5)
f(ev3)
f(ev1)
f(st1)
e([ev4],[r3])e([st2,st3],[r1])
e([st6,ev5],[r1,r2])
e([st7,st8],[r1,r2,r3,r5])
```

```
f(ev2)
f(ev1)
f(st5)
f(ev3)
f(ev1)
f(st1)
e([ev4,st4],[r4])
e([st2,st3],[r1])
e([st6,ev5],[r1,r2])
e([st7,st8],[r1,r2,r4,r5])
```

The dependencies between rules captured in the expansions can be graphically represented by the hypergraphs as depicted in the Fig. 6. 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 (Ramaswamy & Sarkar 1997).

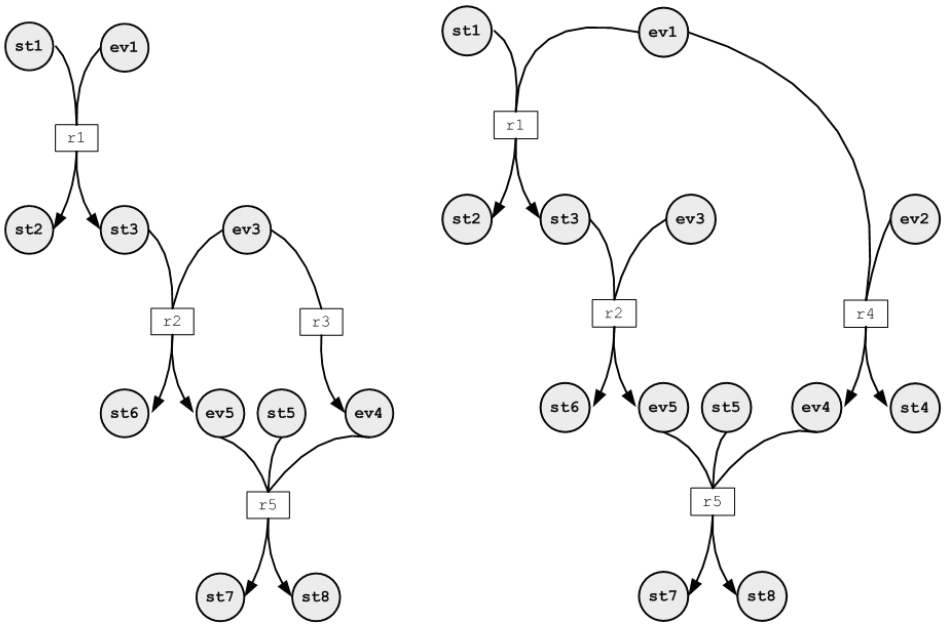

Fig. 6 - Hypergraphs for the literal st8

For sake of brevity, in the described algorithm some simplifications were considered, namely:

- **Local versus global** – as previously referred, in order to fasten the expansion calculation, the knowledge base includes the tuple link/7 that stores the pairs of literals able to mach together. Notice that link/7 assures only a local consistency (between the pair of literals), not global (along the entire expansion). Let's consider the following occurrences of the tuple literalArgs/3:

```
literalArgs(st11,5,[#Dt2,#Tm2,#In1,#Pn1,#_,relRapTrif,closed]).
literalArgs(st11,6,[#Dt2,#Tm2,#In1,#Pn1,#_,relRapMono,closed]).
literalArgs(st11,8,[#_,#_,#In1,#Pn1,#_,#Type,closed]).
```

  In this example, local consistency means that the pairs (st11/5, st11/8) and (st11/6, st11/8) can be matched. Global consistency means the chains (st11/5, st11/8, st11/5) and (st11/6, st11/8, st11/6) can be inferred. Furthermore, the chain (st11/5, st11/8, st11/6) isn't possible, since after the variable Type becomes instantiated with the relTrapTrif, it can't be re-instantiated with the value relTrapMono.
  In order to assure global consistency, the expansion calculation algorithm implements a table of symbols where the variables are stored and updated along with the expansion calculation;
- **Multiple conclusions per rule** – when a literal is expanded if a particular rule infers multiples conclusions that needs to be adequately stored. For instance, the rule r5 allows the inference of the literals st7 and st8, as depicted in the Fig. 6, obviously this situation is reflected in the tuples e/2 and c/2 contained in the expansion list;

- **Data representation –** in the described algorithm the rules are represented using a tuple rule(r,lhs,rhs) although in the algorithm implementation the tuples, previously described, are used (e.g., rule/3, lhs/7, rhs/8 or literal/3).

4.3.2 Temporal analysis

Concerning temporal analysis, the following issues where considered in the VERITAS implementation:

- **Distinct treatment for temporal variables –** this kind of variables are used for labelling literals, hence, during the internal knowledge base creation these variables where stored in lhs/7 and rhs/8 in the field TemporalRef. Later during the expansions calculation, they are indexed in a specific table of symbols with the following structure:

$$(Var, BeginInst, EndInst)$$

  where each variable (Var) has a temporal interval of validity defined by its starting (BeginInst) and ending (EndInst) instants;
- **Capture and evaluation temporal operations –** the literals relating temporal operations contained in an expansion are captured in order to build a net representing their dependencies. This net is later evaluated aiming to assure temporal consistency over an entire expansion. Therefore, the following items are collected:
  - Literals for variables evaluation like: $t = t1 + 1$ and $t = \max(t1, t2)$;
  - Temporal relational operators like: $t1 \prec t2$ and $t \geq 30$;
  - Temporal operators used specifically in SPARSE like:

    `abs_Diff_Less_Or_Equal(t1,t2),` meaning $\|t2 - t1\| \leq t3$.

  Later the collected items are evaluated in order to:
  - Detect inconsistencies between related items like: $t1 \prec t2$ and $t1 \geq t2$;
  - Assert and/or update the table of temporal symbols, for instance, $t1 \prec t2$ and $t \geq \max(t1, t2)$ allow updating variable t with the value of t2.
- **Parametric temporal validity analysis –** the meta-rules stores temporal window of validity for a set of rules. The maximum validity for literals contained in an expansion is defined by the combination of the temporal validity intervals inherited from all rules used in the referred expansion. Additionally, this parametric evaluation is enriched with temporal operations described in the previous items. Regarding that, each literal usually has symbolic, starting and ending instants defined by knowledge base assert and retract operations.

4.3.3 Circularity detection

A knowledge base contains circularity if, and only if, it contains a set of rules, which allows an infinite loop during rule triggering. In order to accomplish modelling requirements, sometimes the knowledge engineer needs to specify rules that allow the definition of a circular inference chain. The algorithm used to compute expansions detects every circular chain existing in a rule base; although aiming to reduce the number of false anomalies, a heuristic was considered for circularity detection. This heuristic has two mandatory conditions:

- At least one literal of the type event needs to be present in the rule LHS. This implies the occurrence of an action requiring some rule triggering. Besides, the referred literal needs to be defined as the triggering fact in the metaRule/7 tuple;
- The culprit literals for circularity (equivalent literals) need to exhibit distinct temporal label. More precisely, the literal that appears in rule RHS needs to occur after the equivalent literal contained in the rule LHS.

4.3.4 Ambivalence detection

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. Concerning the detection of ambivalence, VERITAS is capable of detecting two types: in a single expansion and in multiple expansions.

The detection of ambivalence in a single expansion is performed using an algorithm that works in the following way: for each pair literal1/index1 representing a conclusion (i.e., a literal which appears solely in the rules RHS) contained in the KB the following conditions are evaluated:

- The existence of an expansion that allows to infer the pair literal1/index1;
- The existence of an restriction relating the referred pair;
- The other literal contained in the restriction, literal2/index2, is contained in the expansion considered in the first point.

If all of these conditions are true, it means that two contradictory are contained in the same inference chain. In the last step the validity intervals defined for both literals, represented by literal1/index1 and literal2/index2, respectively, are evaluated, and if they intercept[2] each other then an anomaly is reported.

The detection of ambivalence in multiples expansions is performed using an algorithm that works in the following way: for each constraint contained in KB, if there are expansions supporting both literals contained in the restriction, represented by literal1/index1 and literal2/index2, respectively; finally the algorithm evaluates if the set of literals supporting literal1/index1 contains, or is contained by, the set of literals that support literal2/index2 and if so, an anomaly is reported. Notice that the notion of contain, or contained by, inherited from the set theory is refined with the condition of interception between corresponding literals as depicted in the Fig. 7. The set P (formed by the elements Pi, Pj, Pk and Pl) contains the set Q, since each element of Q exists simultaneously both in P and Q. The set R represents the temporal interception between P and Q.

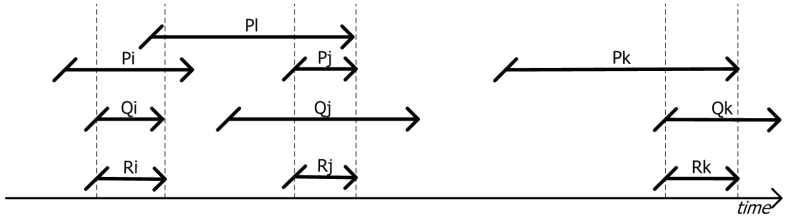

Fig. 7 - Set contains or contained by with temporal characterization

[2] Two temporal intervals intercept each other if they share at least an instant.

4.3.5 Redundancy detection

A knowledge base is redundant if, and only if, the set of final hypotheses is the same in the rule/literal presence or absence. Concerning the detection of the anomaly referred as redundancy, two distinct types where addressed: not usable rule and redundancy in groups of rules.

The detection of unused rules is performed using a rather simple algorithm that works in the following way: for each rule R contained in RB the following conditions need to be verified or an anomaly is reported:

- At least a meta-rule refers the R; if not, the rule wouldn't be ever called;
- The R rule LHS doesn't contain a pair of literals defined in any constraint, as long as an impermissible set of conditions would not be provided as input.

The detection of redundancy in groups of rules is performed using an algorithm that works in the following way: after all expansions calculation, all related expansions for each rule are checked, and if all the conclusions can be inferred by other expansions then the considered rule is redundant. Consequently, an anomaly is reported.

5. Results

In the development of VERITAS some well-known techniques were assembled with a set of new ones specifically designed for VERITAS. The set of techniques referred in literature includes: the calculation of the rule base expansions (Ginsberg 1987); detection of set of anomalies (Rousset 1988; Preece & Shinghal 1994); the use of graph theory for anomaly detection along with inference chains (Preece, Bell & Suen 1992); the use of logical and semantic constraints (Preece & Shinghal 1994; Zlatareva 1991) and directed hypergraphs for representing rule dependencies (Ramaswamy & Sarkar 1997). Therefore, VERITAS has the following characteristics:

- **Independence of the original rule grammar and syntax** - VERITAS includes a module that allows the conversion between original and verification representation formats;
- **Optimized rule base expansions calculation** – in order to fasten the expansions calculation two different techniques were considered: the information needed during the verification process was stored using a normalized data schema in which most important data issues where indexed; the matching pairs of literals were computed *à priori* and stored in the knowledge ensuring *local* consistency;
- **Variables and procedural instructions correctly addressed** – in order to process variables in an adequate way during knowledge base creation, the variables contained in the rule and meta-rule sets are extracted and later stored in the table of symbols. During the expansions calculation, the variables are evaluated and their values are updated in the table of symbols. The use of this mechanism allowed assuring global consistency through an expansion calculation and, consequently, reducing the number of computed expansions. The procedural instructions were considered during expansions calculation, and whenever it implies variables evaluation the table of symbols is updated accordingly;
- **Temporal aspects** – in the development of VERITAS a set of techniques and algorithms were considered in order to address the knowledge temporal reasoning representation issues, namely: definition of an anomaly classification temporal characterized, as well as the temporal characterization of logical and semantic restrictions; variables related

with time representation received a distinct treatment during expansion calculation; capture and evaluation of operations relating time, in order to evaluate the consistency of the net formed by these operations;

- **VERITAS testing** – the verification method supported by VERITAS was tested with SPARSE, an expert system for incident analysis and power restoration in power transmission networks. Regarding that, a previous version of VERITAS was tested with two expert systems for: cardiac diseases diagnosis (Rocha 1990) and otology diseases diagnosis and therapy (Sampaio 1996).

6. Conclusions and future work

This chapter focussed on some aspects of the practical use of KBS in Control Centres, namely, knowledge maintenance and its relation to the verification process.
The SPARSE, a KBS used in the Portuguese Transmission Network (REN) for incident analysis and power restoration was used as case study. Some of its characteristics that mostly constrained the development and use of verification tool were discussed, like: the use o rule selection triggering mechanism, temporal reasoning and variables evaluation, hence, the adopt solutions were described.
VERITAS is a verification tool that performs logical tests in order to detect knowledge anomalies as described.
The results obtained show that the use of verification tools 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.

7. References

Amelink, H., Forte, A. & Guberman, R., 1986. Dispatcher Alarm and Message Processing. *IEEE Transactions on Power Systems*, 1(3), 188-194.

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

Cragun, B. & Steudel, H., 1987. 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.

Duarte, J. et al., 2001. TEMPUS: A Machine Learning Tool. In *International NAISO Congress on Information Science Innovations (ISI'2001)*. Dubai/U.A.E., pp. 834-840.

Faria, L. et al., 2002. Curriculum Planning to Control Center Operators Training. In *International Conference on Fuzzy Systems and Soft Calculational Intelligence in Management and Industrial Engineering*. Istanbul/Turkey, pp. 347-352.

Fisher, M., Gabbay, D. & Vila, L. eds., 2005. *Handbook of Temporal Reasoning in Artificial Intelligence*, Elsevier Science & Technology Books.

Gerevini, A., 1997. Reasoning about Time and Actions in Artificial Intelligence: Major Issues. In O.Stock, ed. *Spatial and Temporal Reasoning*. Kluwer Academic Públishers, pp. 43-70.

Ginsberg, A., 1987. A new aproach to checking knowledge bases for inconsistency and redundancy. *In procedings 3rd Annual Expert Systems in Government Conference*, 10-111.

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

Hoppe, T. & Meseguer, P., 1991. On the terminology of VVT. *Proceedings of the European Workshop on the Verification and Validation of Knowledge Based Systems*, 3-13.

Kirschen, D. & Wollenberg, B., 1992. Intelligent Alarm Processing in Power Systems. *Proceedings of the IEEE*, 80(5), 663-672.

Kleer, J., 1986. An assumption-based TMS. *Artificial Intelligence \(Holland\)*, 2(28), 127-162.

Malheiro, N. et al., 1999. An Explanation Mechanism for a Real Time Expert System: A Client-Server Approach. In *International Conference on Intelligent Systems Application to Power Systems (ISAP'99)*. Rio de Janeiro/Brasil, pp. 32-36.

Menzies, T., 1998. Knowledge Maintenance: The State of the Art. *Engineering Review*.

Nazareth, D., 1993. Investigating the applicability of Petri Nets for Rule Based Systems Verification. *IEEE Transactions on Knowledge and Data Engineering*, 4(3), 402-415.

Nguyen, T. et al., 1987. Knowledge Based Verification. *AI Magazine*, 2(8), 69-75.

Pipard, E., 1989. Detecting Inconsistencies and Incompleteness in Rule Bases: the INDE System. In *Proceedings of the 8th International Workshop on Expert Systems and their Applications, 1989, Avignon,France;*. Nanterre, France: EC2, pp. 15-33.

Preece, A., 1998. Building the Right System Right. In *Proc.KAW'98 Eleventh Workshop on Knowledge Acquisition, Modeling and Management*.

Preece, A., Bell, R. & Suen, C., 1992. Verifying knowledge-based systems using the COVER tool. *Proccedings of 12th IFIP Congress*, 231-237.

Preece, A. & Shinghal, R., 1994. Foundation and Application of Knowledge Base Verification. *International Journal of Intelligent Systems*, 9(8), 683-702.

Ramaswamy, M. & Sarkar, S., 1997. 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*. Leuven,Belgium, pp. 145-154.

Rocha, J., 1990. Concepção e Implementação de um Sistema Pericial no Domínio da Cardiologia. Master Thesis, FEUP, Porto, Portugal.

Rousset, M., 1988. On the consistency of knowledge bases:the COVADIS system. In *Procedings of the European Conference on Artificial Intelligence (ECAI'88)*. Munchen, pp. 79-84.

Sampaio, I., 1996. Sistemas Periciais e Tutores Inteligentes em Medicina - Diagnóstico, terapia e apoio de Otologia. Master Thesis, FEUP, Porto, Portugal.

Suwa, M., Scott, A. & Shortliffe, E., 1982. An aproach to Verifying Completeness and Consistency in a rule based Expert System. *AI Magazine (EUA)*, 3(4), 16-21.

Vale, Z. et al., 1997. SPARSE: An Intelligent Alarm Processor and Operator Assistant. *IEEE Expert, Special Track on AI Applications in the Electric Power Industry*, 12(3), 86-93.

Vale, Z. et al., 2002. Real-Time Inference for Knowledge-Based Applications in Power System Control Centers. *Journal on Systems Analysis Modelling Simulation (SAMS), Taylor&Francis*, 42, 961-973.

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

Wu, C. & Lee, S., 1997. 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.

Zlatareva, N., 1991. Distributed Verification: A new formal approach for verifying knowledge based systems. In I. Liebowitz, ed. *Proceedings Expert systems world congress*. New York, USA: Pergamon Press, pp. 1021-1029.