Computación y Sistemas Vol. 8 Núm. 4, pp. 265-280
© 2005, CIC-IPN, ISSN 1405-5546, Impreso en México

CRIB: A Method for Integrity Constraint Checking on Knowledge Bases

CRIB: Método para la Comprobación de Restricciones de Integridad en Bases de Conocimiento

Julia Clemente[1], Angélica de Antonio[2] and Jaime Ramírez[2]

[1]Universidad de Alcalá de Henares
Escuela Universitaria Politécnica
Departamento de Automática
Campus Universitario. Ctra. Madrid-Barcelona, Km. 33,600
28871 Alcalá de Henares, Madrid, Spain
e-mail: julia@aut.alcala.es
[2]Facultad de Informática.
Universidad Politécnica de Madrid.
28660 Boadilla del Monte, Madrid, Spain.
e-mail: {angelica, jramirez}@fi.upm.es

Article received on october 15, 2003; accepted on february 28, 2005

Abstract

The necessity of verification tools for Knowledge Based-Systems (KBSs), that help to guarantee a certain degree of quality and reliability of these systems will increase in the future when more critical systems are developed in areas such as industry, science, business, etc. One of the objectives of the KBSs verification is to assure the consistency and completeness of the Knowledge Base (KB). In this paper, a technique to detect possible inconsistencies or conflicting situations between the objects of the KB is described, and a tool called CRIB, that implements this technique, is presented. The generality of this technique, based on the checking of the Integrity Constraints (ICs) declared on the KB, will allow to apply it to different kinds of KBSs. In addition, the flexibility and the power of the ICs specification language will make it possible to detect a wide range of inconsistencies in a KB.
Keywords: Verification, Knowledge Based-System, Consistency.

Resumen

La necesidad de herramientas de verificación para Sistemas Basados en el Conocimiento (SBCs), que ayuden a garantizar un cierto grado de calidad y fiabilidad de estos sistemas aumentará en el futuro conforme más sistemas críticos sean desarrollados en áreas tales como la industria, la ciencia, los negocios, etc. Uno de los objetivos de la verificación de SBCs es asegurar la consistencia y la completitud de la Base de Conocimientos (BC). En este artículo se describe una técnica para detectar posibles inconsistencias o situaciones conflictivas entre los objetos de la BC, y se presenta una herramienta llamada CRIB, que implementa esta técnica. La generalidad de esta técnica, basada en la comprobación de las Restricciones de Integridad (RIs) declaradas en la BC, permite aplicarla a diferentes tipos de SBCs. Asimismo, la flexibilidad y la potencia del lenguaje de especificación de RIs hará posible detectar un amplio abanico de inconsistencias en una BC.
Palabras Clave: Verificación, Sistema Basado en el Conocimiento, Consistencia.

1 Introduction

From the beginning of the decade of the 1980s Knowledge-Based Systems (KBSs) are being frequently used in different areas such as industry, science, defense, etc., mainly in complex and large applications.

The future growth of KBSs will be seriously obstructed if developers do not firmly face the problem of reliability. The trustworthiness of KBSs in critical use, such as diagnostic systems, nuclear power plant management systems, spatial vehicles control systems, etc., and of many other systems for which the use is not as critical as in the previous ones (like manufacturing process control systems, traffic control systems, and so on), must be guaranteed before they can be used for these tasks. In addition, loss of credibility may come from fault conclusions derived from the system. This could lead to a decrease in the use of such systems, and finally, to their final retirement.

One of the main components of all software development projects is, of course, to check it out as a part of the gradual development process. However, besides the elimination of mistakes effected during the development, a formal evaluation of KBSs should be performed. Verification and Validation (V&V) of such systems provide the means and mechanisms to effect the necessary formal evaluation.

Although the distinction between Validation and Verification has not been yet fully clarified [1] [2] [3], the main objective of both processes is to guarantee that the KBS provides the correct answer in the proper way when it is required to solve a problem. In addition, both terms also imply other goals such as to assure the maintenance, security and service of the system. To achieve this, any mistake or deficiency in the system must be first discovered and then eliminated.

1.1 Verification and Validation

The term "*Verification and Validation*" (V&V) is often used to indicate all the issues related to assuring that the software will perform the task which it was designed for. In this sense, the term V&V is used as a rough synonym of "evaluation".

There are many definitions of the term "Verification" in the literature [4] [5] [6]. The following definition is considered in this paper:

*The **Verification** confirms that the expert system is logically consistent, but it does not guarantee that its domain-dependent knowledge agrees with that of the human expert (Suen et ál., 1990).*

When the KBSs are being verified, their main components must be considered, i.e., their Inference Engine (IE), and their Knowledge Base (KB). Conventional software verification procedures must be applied to the IE since this component is a conventional software piece. On the other hand, according to the VALID project (Cardeñosa & Juristo, 1993), the verification of the KB is composed of two steps: the detection of syntactic and semantic mistakes in the KB (*Structural Verification* and *Verification of the content* respectively).

Regarding the term "Validation", the literature about KBSs provides several definitions [4] [5] [7] [2]. We will adopt the following definition:

*The **Validation** is the process of evaluating an expert system during and after the development process to ensure compliance with the initial requirements (Suen et ál., 1990).*

1.2 Inconsistency Detection

One of the activities that should be performed during the Verification of the KB contents is *to prove the consistency of the KB*. The current literature provides too many variants about the meaning of the term *consistency* [8] [9] [10] [11] [12]. We consider that this term implies not only the logical but also the semantic aspects. Thus, a KB is *inconsistent* when it is possible to reach conflictive situations among the KB objects from a set of valid input data.

A set of inputs is *valid* if it represents a situation that is possible in the real world and, therefore, it verifies all the defined integrity constraints.

The type of conflicting situations that can appear when a KB is inconsistent depends on characteristics of the KB representation formalism [12].

1.3 Overview of the Paper

In this paper we approach the problem of inconsistency detection in a KB. In section 2, we present a short history of some tools that have been built to solve this problem. With the goal of solving some of the limitations of the methods used in these tools, we present in this paper a solution that tries to improve them. It is based on several characteristics of some of those methods, but providing new ideas.

In section 3, we will describe the hypotheses adopted in the design of the CRIB method presented in this paper. In the next section, we give a definition and classification of the integrity constraints that will be considered on the KB. Finally, the CRIB method is explained.

2 Previous Work in this Area

With the aim of detecting inconsistencies in KBs based on monotonic reasoning, several methods have been proposed to date. These methods can be grouped into different families, according to the approach they follow. We will review these families now:

- *Tabular Methods* [13] [14]: tabular methods were the first ones to be designed for detecting anomalies in KBSs. The strategy followed by a tabular method consists on comparing in pairs the rules of the rule base, with the objective of discovering certain relationships among their premises and conclusions. Given that these methods only compare pairs of rules, all the inconsistencies in which chains of several rules are involved are beyond their scope.
- *Methods based on Petri Nets* [15] [16] [17] [18]: the methods in this family are based on modeling the KB by using a Petri net, and applying techniques of diverse nature to the Petri net with the aim of detecting inconsistencies. These methods

need to test the model under all the possible initial states in order to warrant the absence of inconsistencies, which can be computationally very costly.

- *Methods based on Graphs* [19] [20] [21] [22] [23] [24] [25]: graphs are a very attractive tool in order to represent conceptual dependencies. Moreover, graph theory provides us with analysis techniques to study properties such as conectivity or reachability in a rigorous and formal way. On the other hand, the applicability of methods based on graphs, as it happens with methods based on Petri nets, can be formally proved. However, they have as a disadvantage, like Petri net-based methods, the necessity of simulating the execution of the system for every possible initial fact base.
- *Methods based on the Generation of Labels* [8] [9] [26] [12] [27] [28] [29]: these methods are grounded on the ATMS designed by de Kleer [30]. Given that many times the generation and propagation of every possible initial FB is computationally intractable, these methods take the concept of *context*, used in the ATMS, as a means of specifying a set of initial FBs. Contexts that describe initial FBs which cause the deduction of an inconsistency will be specially interesting. Therefore, the goal of these methods will be the construction of those contexts. As the number of rules in the RB is finite, the specifications contained in a context will also be finite, even if the number of possible initial FBs is infinite.
- *Methods based on Algebraic Interpretations* [31] [32] [33]: these are systems that are based on the transformation of the KB into an algebraic structure, like a boolean algebra, and the application of concepts and procedures relative to that algebraic structure for the verification of the KB. This family of methods is the most reliable, since they are solidly grounded on algebraic concepts. However, they can only be applied to propositional rule bases.

There are not many methods that are able to deal with the verification of non-monotonic RBs. Among them, we can highlight Antoniou's method [34], which analyzes RBs expressed in default logic, and Wu & Lee's method [35], which tests RBs with production rules. While Antoniou's method is inspired on tabular methods for monotonic RBs, Wu & Lee's method models the RB as a high level extended Petri net.

Analogously, only a few methods have been proposed that are able to verify systems with uncertainty management [36] or hybrid systems. The first work that dealt with hybrid systems was that of Lee & O'Keefe [37], which characterized a set of new types of anomalies that appear as a result of considering the submission relationship among literals in different rules. The submission relationship is a consequence of the *subclass-of* relationships that appear in a frame hierarchy. In order to detect those anomalies, Lee & O'Keefe presented a method with a tabular approach. Later on, as an extension of Lee & O'Keefe's work, Mukherjee et ál. [38] proposed a similar procedure to detect dead-end and unreachable rules, considering the submission relationship. They also proposed a method based on the generation of labels to detect absent rules. Mukherjee et ál. also introduced an algorithm to detect anomalies caused by the interaction of rules and monitors (methods and daemons) during the execution of the KBS.

The CRIB method presented in this paper is inspired on the KB-REDUCER [9] and TADIKS [26] tools based on the generation of labels. However, there are great differences in theis starting points as well as in the implementation. The improvements that CRIB introduces in contrast to those previous tools are essentially: more extensive scope, detection of a wider range of possible inconsistencies in the KB independently of the initial fact base (FB), providing more information to correct the potential inconsistencies in the KB, and allowing an incremental verification. These characteristics are explained in the section 3.

3 Hypotheses and Goals of CRIB

The main goals and hypotheses behind CRIB are the following:

* *Wider scope*: CRIB has been designed to be integrated in the VALID validation environment, product of the VALID project [39] which will allow the tool to be applied in different kinds of KBSs.

In the VALID environment, a generic representation kernel called CCR (Common Conceptual Representation) and a validation meta-language, called VETA (Validation mETAlanguage) [40], are included. Therefore, CRIB will be applied to KBSs represented in CCR and it will be implemented using the set of primitives of VETA.

The generality of the CCR representation formalism, as we will show below, allows to extend the scope of the existent tools. CCR allows the representation of Attribute-Value pairs, Object-Attribute-Value triplets, classes and instances of classes, and to organize them in hierarchies of classes. Besides, CCR supports a modular knowledge representation that allows to represent the rule structure of the KB, and some mechanisms to control the execution of the KBS, like metarules.

Regarding the limits in the CRIB scope, those are essentially determined by the limits in the CCR representation capability. In this way, for example, procedural knowledge is not considered in the CRIB tool since CCR cannot represent the semantics related to it.

Most of the methods to detect anomalies in the content of the KB are applied only to such knowledge base, without taking care of the inferential and control aspects. This is the case of KB-REDUCER and TADIKS. But, many times, the accuracy of the results obtained will depend on the KBS inference engine properties. Though CCR cannot represent the

semantics related to the conflict set resolution criteria, the importance of simulating the behavior of the most used conflict set resolution criteria has been considered in the design of CRIB. The user of CRIB will be able to select, among these criteria, the one which the KBS that is being verified is actually using. In contrast, control knowledge mechanisms and metarules will not be considered, although some of those can be represented in CCR.
* *Detection of a wider range of possible inconsistencies in the KB*: The KB-REDUCER and TADIKS tools only detect *semantic and logic incompatibilities* and *incompatibilities on the cardinality of an attribute* (see section 4) expressed in propositional logic, but they don't detect the violation of other constraints that can be defined on the KB. CRIB allows to represent consistency criteria by using IC. The IC specification language used in CRIB (we will describe it further in section 5) is characterized by its flexibility and clarity.
* *Analysis of the potential inconsistencies in the KB independently of the initial fact base*: This is the key idea which has inspired the design of CRIB and which makes the difference with many existing tools. CRIB analysis is not limited to specific executions of the KBS with particular cases named initial fact bases (FBs). Therefore, it will allow to detect all the potential inconsistencies in the KB for all the valid inputs to the system. This feature will facilitate enormously the work of the user of CRIB, who will not have to execute the tool for each specific initial FB, obtaining in this way more general and helpful information.
* *Provide information to correct the potential inconsistencies in the KB*: CRIB offers more information than previous tools like KB-REDUCER and TADIKS, that will help the user or knowledge engineer to find the root causes of the potential inconsistencies in the KB.

4 Integrity Constraints

The method we describe in this paper is based on the declaration and checking of ICs defined on a KB in order to verify its consistency. Consistency (or inconsistency) criteria should therefore be formulated as integrity constraints.

*An **integrity constraint** on a KB is defined as a set of conditions that the knowledge contained in the base must satisfy to be considered a consistent model of the real world.*

*The **integrity constraint satisfaction** is defined as the fact of giving strict fulfilment to the conditions imposed by the constraint.*

*The **integrity constraint violation** is defined as the fact of not giving strict fulfilment to the conditions imposed by the constraint.*

The causes that lead to an IC violation and, therefore, to inconsistent situations in the KB, derive from the application of the knowledge contained in the KB, in order to solve a specific problem (defined by an initial FB), by using the available inference mechanisms in the system. Some factors that are going to have considerable influence on the structure and satisfaction or violation of the ICs imposed on a KBS are the following:

a) Representation formalism(s) used in the KBS (rule-based representation, semantic nets, frames, etc.): The violation of an IC defined on the KB can be produced not only by the firing of rules, but also by the matching in semantic nets or frames, by the inheritance of attribute values, by the execution of a method or daemon or by any combination of them.

b) Type of underlying logic in the system (propositional logic, predicate logic) and the way to represent logical facts (Attribute-Value pairs, Object-Attribute-Value triplets, etc.): According to the kind of logic, the type of facts we handle in the KBS and in the ICs will be different.

c) Uncertainty management mechanism: In systems with uncertainty management, the presence of conflicts may not be problematic, if the expert deliberately concludes contradictory facts with different certainty values. The next figure shows the classification of ICs that is used in CRIB:

a) ***Incompatibilities***: ICs that express incompatibilities may appear between facts or between attribute values.

a.1) ***Incompatibilities between facts***: The constraints that define incompatibilities between facts may express *logic* or *semantic incompatibilities between facts.*

- ***Logic Incompatibilities***: They denote an assignment of truth values to facts in such a way that a fact and its negation are simultaneously considered true. Example (propositional logic): H and $\neg$H.
- ***Semantic incompatibilities***: They denote an assignment of truth values to facts in such a way that a set of facts that are semantically incompatible are simultaneously considered as certain. Example (first order logic): Is_man(a) and Is_inmortal(a).

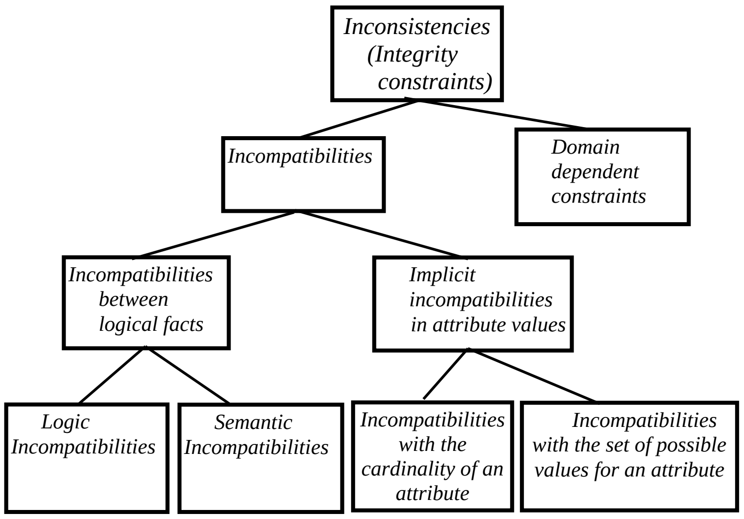
Inconsistencies (Integrity constraints)
Incompatibilities
Domain dependent constraints
Incompatibilities between logical facts
Implicit incompatibilities in attribute values
Logic Incompatibilities
Semantic Incompatibilities
Incompatibilities with the cardinality of an attribute
Incompatibilities with the set of possible values for an attribute

Fig. 1. ICs classification

a.2) ***Implicit incompatibilities in attribute values***: They may appear when the fact representation formalism in the system is of the type Attribute-Value or Object-Attribute-Value. Two types of incompatibilities on the values of the attributes are possible:

- ***Incompatibilities with the cardinality of an attribute***: Some attributes of the KB may have related constraints on the minimum or maximum number of values that they may take in any moment. Example (Attribute-Value) : Weight = 70 and Weight = 80 is an incompatibility because the Weight attribute may only have one value.
- ***Incompatibilities with the set of possible values for an attribute*** : This kind of incompatibilities derive from the typing of the attributes, i.e., from the specification of a set of possible values. Example (Object-Attribute-Value): If (Man,Weight,300) is in the FB and it has been declared that Weight $\in$ (1,150] then, (Man,Weight, 300) is an incompatibility.

b) ***Specific Domain dependent constraints***: Any constraint that implies specific conditions or characteristics which must be satisfied by the different objects of a particular KB and that cannot be included in the previous classification, such as conditions on the existence of the objects or relationships between them, belong to this group of ICs. Example: "*All the children of a marriage must be boys or girls and their age must be known*". This IC is only applicable if a marriage has children. In that case, they must belong to the *boy* or *girl* class (not both) and, besides, their age must be known.

5 CRIB Method

We begin this section with a short description, by using examples, of the ICs specification language used by CRIB and then we will explain the method proposed to solve the inconsistency detection problem.

5.1 The ICs specification Language

The ICs specification language used by CRIB is based on the language of the CONTROLLER tool [41], although there are some differences, mainly in its syntax, which has been simplified.

5.1.1 Facts Representation

In the CRIB method, a fact is a predicate applied to a KB object: classes, instances or instance attributes. For example[1]:

Notes.
1. All the examples we show in this paper are represented with a simplified notation for the sake of clarity.

1) ≤ ((car1,speed) , 200): It represents a fact that is true if the attribute *speed* of the instance *car1* is lower or equal (predicate ≤) than 200.
2) <>((client1,age), 40): It represents a fact that is true if the attribute *age* of the instance *client1* of the class *client* is different (predicate <>) from 40.

In addition, it is possible to represent facts using *meta-predicates* or special predicates that contain high-level knowledge about the KB objects. Example: IS_OF_CLASS (customer, John). This fact is true if *John* is an instance of the class *customer*.

5.1.2 Scope of ICs

The constraints expressed in this language will belong to one of the following types, depending on their *scope*:

* ***Constraints that only deal with input data***: These constraints establish consistency criteria that the KBS input data must satisfy.
* ***Constraints that only deal with output data***: These constraints establish consistency criteria that must be satisfied by the system output data. The initial FB must satisfy the KB input constraints (valid initial situation).
* ***Constraints that deal with input and output data***: These constraints establish consistency criteria that must be satisfied by the KB input data as well as by the KB output data.

5.1.3 The IC Structure

The main grammatical rules that define the structure of an IC are the following. They are expressed in EBNF (Extended Backus Naur Form) notation.

```
< Constraint >::= < Constraint_Identification >
                  < Scope_Specification >
                        [ < Quantifications > ]
                        < Constraint_Condition_Part >
                        < Constraint_Body >
< Constraint _Condition_Part > : :=  "IF" < Constraint_Condition >
< Constraint_Condition > : := ( < Local_Quantifications > )*    "(" ( < Factors > ) ")"
< Factors > : :=  < Factor > |
   "(" < Factor > ")" "OR" < Other_Factors >
< Other_Factors > : :=  "(" < Factor > ")" |
   "(" < Factor > ")" "OR"  < Other_Factors >
< Factor > : := < Conditions > |
   ( < Local_Quantifications > )* < Conditions >
< Conditions > : :=  "(" ("NOT")  < Condition > ")" |
    "(" ("NOT")  < Condition > ")" <Conditions >
< Condition> ::=  <Language_Clause>  |
                        <LISP_Clause>
< Constraint _Body > : :=  { "THEN" | "THEN _NOT" }   "(" < Factor > ")"
```

In the following, the semantic of these structure fields is shortly explained:
<Constraint_Identification>: It is the identifier of the constraint.
<Scope_Specification>: It allows to define the type of constraint depending on its scope (see section 5.1.2).
<Quantifications>: This field is optional. It allows to define existentially or universally quantified variables that appear in both IC parts (condition part and IC body).
<Constraint_Condition_Part>: Its aim is to filter the cases in which the *constraint body* must be evaluated. This part is composed of one or several sets of conditions named *factors*. The factors are separated from each other by the OR operator and the factor conditions are implicitly separated by the AND operator. Each condition may be preceeded by the NOT operator.
<Constraint_Body>: It is composed of only one factor.
Each factor, condition part or constraint body can be also preceded by quantified variables.

<Condition>: The conditions can use language predicates to evaluate attribute values (e.g: BELONGS-TO, IS-DIFFERENT-FROM, IS-EQUAL-TO, IS-GREATER-THAN, IS-LESS-THAN, etc) or language metapredicates to check if an instance belongs to a class (IS-OF-CLASS) or does not belong to it (IS-NOT-OF-CLASS), metapredicates to check if a class has instances (HAS-INSTANCES) or not (IS-EMPTY), etc. A condition can also use an S-expression of LISP with predefined or user functions.

In section 5.1.2 we saw that the constraints can be classified according to their scope. Besides, depending on the *satisfiability of the IC body*, the constraints can belong to one of the following types:

* ***Positive constraints***: If their condition part is satisfied, the constraint body must also be satisfied in order to conclude the IC satisfaction.
** **Negative constraints***: If their condition part is satisfied, the constraint body should not be satisfied in order to conclude the IC satisfaction.

If the constraint condition part was not satisfied, its body wouldn't be evaluated.

5.1.4 Examples

Example 1: Suppose that, between the KBS input and output the following constraint has to be satisfied: "*It is not possible to apply two different allowances to the same child*".

This IC may be written in the CRIB language as:

```
CONSTRAINT ALLOWANCE_CHILD
SCOPE (IO)
WHATEVER *CHILD (I)
IF ( )
THEN_NOT
THERE_IS (*1ALLOWANCE (O) *2ALLOWANCE (O) )
( (THE reason OF *1ALLOWANCE  IS_EQUAL_TO  *CHILD) (THE reason OF *2ALLOWANCE IS_EQUAL_TO   *CHILD) )
```

The words CONSTRAINT, SCOPE, WHATEVER (it represents the universal quantifier), THERE_IS (it represents the existential quantifier), IF, THE, OF and THEN_NOT are language keywords. The constraint is identified as ALLOWANCE_CHILD and it is an input and output constraint (this is expressed by SCOPE (IO)). The THEN_NOT keyword specifies that it is a *negative constraint*. The condition part, preceded by the IF keyword, will always be true because it has been defined as ( ).

The constraint will be evaluated for each input instance of the class CHILD. *CHILD, *1ALLOWANCE and *2ALLOWANCE are variables that respectively take as values instances of the CHILD and ALLOWANCE classes. The attribute *reason* will be evaluated in the output instances of the ALLOWANCE class((O)).

Example 2: Suppose that, in the KBS input, the following constraint has to be satisfied: "*A woman whose father is unknown must be named as her mother* ".

This IC may be written in the CRIB language as:

```
CONSTRAINT SURNAME
SCOPE (I)
WHATEVER *PERSON (I)
IF
THERE_IS *1WOMAN
  ( (THE sex OF *PERSON IS_EQUAL_TO female)
  (THE father OF *PERSON IS_UNKNOWN)
  (THE mother OF *PERSON IS_EQUAL_TO *1WOMAN)
  (THE surname OF *1WOMAN IS_EQUAL_TO ?surname_mother) )
THEN
 (  (THE surname OF *PERSON  IS_EQUAL_TO  ?surname_mother) )
```

The THEN keyword specifies that this is a *positive constraint*. IS-UNKNOWN is a language metapredicate to check if an attribute is valuated. The remaining keywords were previously explained in the example 1.

The constraint is an input constraint (this is expressed by SCOPE (I)). The variable *?surname_mother* is called an *attribute variable* because it takes as its value the value of the *surname* attribute. In addition, this variable appears in the constraint body where it is instantiated by propagation, because the *?surname_mother* value has already been established before in the condition part.

5.2 The Method

In this section, we are going to explain what the CRIB method of inconsistency detection consists of. Beforehand, we will define the concept of *context* that the method manages. Next, we will explain some operations on contexts. Finally, we will describe how the method operates.

5.2.1 Context

The objective of this method is to obtain an specification of all the initial FBs, and the rules that have to be fired (and in which order) from these initial FBs in order to produce the violation of an IC. A set of initial FBs will be described by an *environment*, whereas a list of rules will be described by a *deductive path.*

* An ***environment*** E_i represents the specification of the set of all valid initial FBs in which, at least, all the facts included in E_i are certain.
* A ***deductive path*** DP_i associated with an environment E_i is defined as a list of rules whose firing in the specified order is possible, given the facts included in E_i , and lead to the satisfaction of a given goal.
* A ***context*** C represents a set of pairs (E_i, P_i) such that:

$C = \{ (E_i, P_i); i=1,...,n \}$
where E_i represents an environment and
$P_i = \{ DP_1 , ... , DP_m \}$ is the set of deductive paths associated with E_i .

Contexts can be associated with facts, rules and integrity constraints. An example context could look like:

{ ({a,b,d}, { (R2,R5), (R3,R1,R6) }),
({a,f,g,s}, { (R5,R4) }) }

and {a,b,d,m,t}, {a,b,f,g,h,p,s} would be initial fact bases that satisfy this context.

If *C* is a context associated with a fact *h*, then any pair *(E, DP)* contained in *C* satisfies the following proposition: after firing the rules included in *DP* from *E*, the fact *h* will be true. If *C* is a context associated with a rule *r*, then, after firing the rules included in *DP* from *E*, the rule *r* could be fired.

If the context associated with an IC is empty, that means that such constraint is always satisfied for all valid initial FBs. Otherwise, if the context associated with an IC is not empty, then each pair included in the context represents a set of valid initial FBs that lead to the IC violation during the KBS execution.

Thus, the objective of the method can be reformulated as computing the context associated with all the ICs declared on the KB, and pointing out an inconsistency whenever a non-empty context is calculated.

5.2.2 Operations on Contexts

The following operations are used during the context computing process:

a) ***Context creation***: This operation is used when the method has to compute the context associated to an external fact. The context associated with the external fact *h* is equal to the pair *(E, DP)* where $E = \{h\}$ and $DP = \varnothing$. An external fact is a fact that cannot be deduced by the KBS and therefore it has to be provided as an external input in order to be true.
b) ***Combination of a pair of contexts***: Let C_1 and C_2 be two contexts. Then each pair belonging to C_1 is combined with each pair belonging to C_2. In this way, if C_1 contains n_1 pairs and C_2 contains n_2 pairs, the resulting context will contain $n_1 x n_2$

pairs. The form of each pair belonging to the resulting context will be *($E_i \cup E_j$, $P_i * P_j$)* where *(E_i, P_i)* is a pair contained in C_1 and *(E_j, P_j)* is a pair contained in C_2.

b.1) ***Union of environments ($E_i \cup E_j$)***: this operation consists of the union of the sets E_i and E_j. If the environment resulting from this union is a superset of any environment included in the context associated to an input IC, then the pair will be discarded, since it would correspond to an invalid initial situation. Otherwise, the resulting environment is considered consistent, and the following operation is executed:

b.2) ***Combination of sets of deductive paths ($P_i * P_j$)***: Let P_i and P_j be two sets of deductive paths. Then each deductive path belonging to P_i is combined with each deductive path belonging to P_j. The result of the combination of two deductive paths will be another deductive path.

b.3) ***Combination of deductive paths ($DP_s * DP_t$)***: This operation consists of two steps:

1.Obtain DP_t' from DP_s and DP_t so: a certain rule *r* will belong to DP_t' if and only if r belongs to DP_t but r does not belong to DP_s.

2.Append lists DP_s and DP_t'.

Before combining the two deductive paths, it needs to be tested if there is a pair of contradictory actions *A* and *B*, such that *A* belongs to DP_s and *B* belongs to DP_t. Two actions are contradictory if they assign a different value to the same pair (object, attribute). If there is a pair of contradictory actions in DP_s and DP_t, then these deductive paths will not be combined.

Let us look at an example of context combination:

C_1 = (E_1, P_1) with
E_1 = { Exists (John), ≥ ((John, allowances),15) }
P_1 = { (R1), (R3, R4, R7) }

C_2 = (E_2, P_2) with
E_2 = { Is-Of-Class (John, customer) }
P_2 = { (R1), (R7, R8) }

The pair resulting from the (E_1, P_1) and (E_2, P_2) combination will be (E_r, P_r) with:

E_r = { Exists(John), ≥ ((John, allowances),15), Is-Of-Class (John, customer)}
P_r = { (R1), (R1,R7,R8), (R3,R4,R7,R1), (R3,R4,R7,R8) }

c) ***Concatenation of a pair of contexts***: The concatenation of contexts consists of the union of all the pairs belonging to the contexts to which this operation is applied.
Example:

C_1 = (E_1, P_1) with
E_1 = { Exists (customer1), = ((customer1, income),high) }
P_1 = { (R1) }

C_2 = { (E_2, P_2) , (E_3, P_3) } with
E_2 = { > ((customer1,benefits), 1) , = ((operation, period),10) } P_2 = { (R2) }
E_3 = { ≥ ((customer1, allowance),15) }
P_3 = { (R1,R3), (R4) }

the context resulting from the C_1 and C_2 concatenation will be:
C_r = { (E_1, P_1), (E_2, P_2), (E_3, P_3) }

After combining or concatenating a pair of contexts, it may happen that repeated pairs appear in the resulting context. These repeated pairs should be removed from the resulting context.

5.2.3 Operation of the Method

In the following figure the general scheme of CRIB execution is shown.

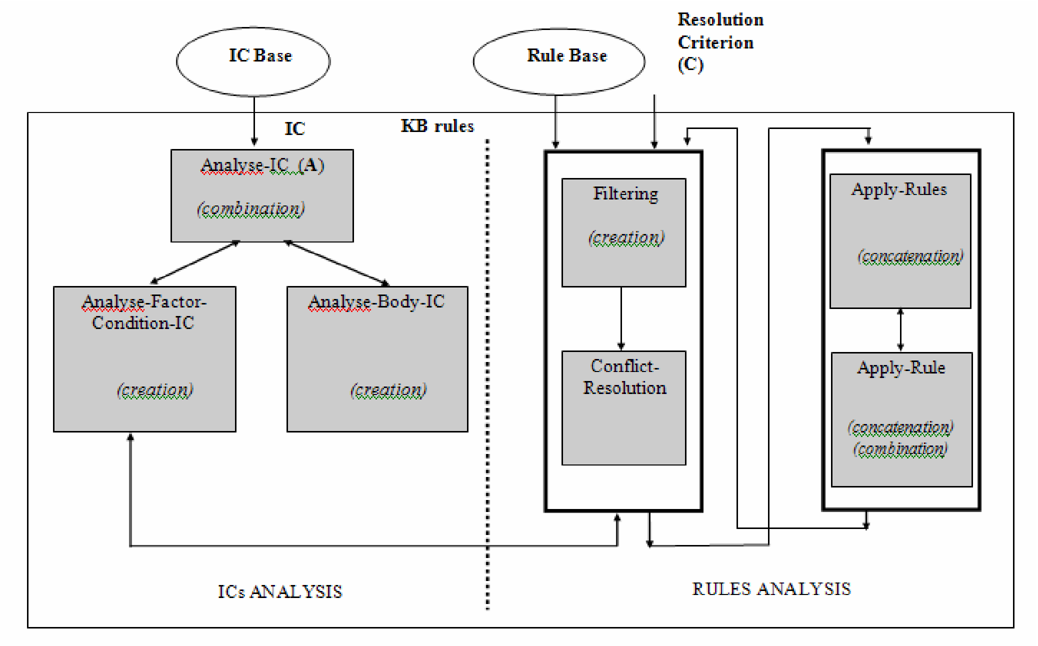

Fig. 2. General Scheme of CRIB execution

This method consists of a process that allows the user, not only to analyze a given set of constraints (IC Base in the figure 2), but also to add, modify or delete constraints and to study the effect of this change without having to repeat the whole process. The analysis is performed in an incremental fashion.

The information that the method provides to the user is not restricted to an indication of consistency or inconsistency. Additional valuable information is provided to help the user to find the malformations in the KB that lead to an inconsistency: name and status (not applicable/satisfied/violated) of each of the selected constraints and, for each violated constraint: the instantiated conditions, both in the condition part and the body part, that lead to the violation of the constraint, together with the associated context.

For each IC selected by the user, the following process is executed:

Process to obtain the context associated to an IC

```
1.Instantiate IC’s variables.
2.Obtain the context associated to the IC’s condition part.

     2.1.FOR each factor in the condition part:
       2.1.1.FOR each condition in the factor:
         2.1.1.1.Obtain the context associated to the condition.
       2.1.2.Combine contexts associated to conditions.
     2.2.Concatenate contexts associated to factors to obtain the context associated to the IC’s
     condition part.
3.IF context associated to condition part is not empty
  THEN (* IC is applicable *)
     3.1.Obtain the context associated to the negation of the IC’s 3.2.Combine the context associated
     to the IC’s condition part and to the negation of the IC’s body to obtain the context associated
     to the IC.
  ELSE (* IC is not applicable *)
     3.3.The context associated to the IC is empty.
```

Since the IC structure is $\Theta_1 x_{1^o},\Theta_2 x_2,...,\Theta_n x_n$ *(A (condition part)* $\Rightarrow$ *B (body part))*, where Θ_i represents either a universal or an existential quantification, and x_i is a variable, this process is going to obtain the context associated to the formula $\Theta_1{'} x_{1^o},\Theta_2{'} x_2,...,\Theta_n{'} x_n$ $(A \wedge \neg B)$, where Θ'_i represents the contrary quantification to Θ_i, i. e. if Θ_i is a universal quantification, then Θ'_i is an existential quantification, and vice versa. Hence, every set of valid inputs which lead to the deduction of this formula will permit the violation of the IC.

In the first step, each IC's variable is bound to a generic constant. As the IC's variables can have universal quantification or existential quantification, the generic constant will inherit the variable's quantification. For example, if variable's quantification is universal, the generic constant will make reference to any object in the FB.

In order to generate the context associated to the IC, it is necessary to obtain the context associated to the condition part and the context associated to the negation of the body part (steps 2 and 3.1), and to combine them.

The condition part contains a list of factors joined by disjunction operators; therefore the context of each factor must be obtained (step 2.1), and then they must be concatenated (step 2.2). Each factor contains a list of conditions joined by conjunction operators; thus in order to obtain the context of a factor, the context of each condition must be generated (step 2.1.1.1).

If the context of the condition part is empty, the IC is not applicable. Hence, there is not a context associated to the IC. On the other hand, if the IC is applicable, the context of the negation of the body part has to be obtained (step 3.1).

Process to obtain the context associated to a condition

```
1. Filter rules that confirm the condition (goal) ⇒ conflict set.
2. IF conflict set is empty
   THEN (* the condition is an external fact *)
       2.1. Create context associated to the condition.
          ELSE (* the condition is a deducible fact *)
       2.2. Sort the conflict set's rules according to the criterion selected by the user ⇒ conflict
       list.
       2.3. FOR each rule in the conflict list:
          2.3.1.Obtain the context associated to the rule.
       2.4. Concatenate the contexts associated to all the rules included in the conflict list.
```

In the process to obtain the context of an IC, we can see that it is necessary to generate contexts associated to all the conditions included in the IC. Then, in order to generate the context of a condition, if the condition refers to a deducible fact, the contexts associated to all the rules which allow to deduce the condition must be obtained (step 2.3 in the previous process).

The rules obtained in the filtering are sorted according to a criterion selected by the user (*C* in the figure 2). In this way, the conflict set is converted into the conflict list (step 2.2 in the previous process). Next, the contexts of the rules are obtained in the same order in which they appear in the conflict list. Finally, contexts are concatenated to generate the context associated to the initial goal (step 2.4 in the previous process).

Process to obtain the context associated to a rule

```
1. FOR each conjunction in the rule premise:
     1.1.FOR each condition of a conjunction:
         1.1.1. Obtain the context associated to a condition.
     1.2. Combine contexts associated to every condition included in the conjunction.
2. Concatenate contexts associated to every conjunction included in the rule premise.
```

Obtaining the context associated to a rule implies calculating the context of every conjunction included in the rule premise (step 1 in the previous process). Each conjunction contains a list of conditions. Therefore, in order to calculate the context of a conjunction, the context of each condition included in the conjunction must be obtained (step 1.1 in the previous process). Then, the context of the conjunction will be the combination of the contexts associated to every condition (step 1.2 in the previous process), and the context of the rule will be the concatenation of the contexts associated to every conjunction (step 2 in the previous process).

These three processes represent a simulation of the firing of rules that occurs when the KBS is executed.

The recursive calls finish, when in the process of obtaining the context of a goal, the goal is an external fact.

The following example will show the application of the method with a positive constraint. For the sake of clarity a simplified notation is used.

Constraint EXAMPLE 1

$\forall x_{CUSTOMER}\ \exists y_{OPERATION}$

$=((x, \text{client-assessment}),\text{interesting}) \wedge \geq((y,\text{period}),8)) \vee$

$>((y,\text{volume}),100) \rightarrow$

$=((y,\text{factor-recommendable}),\text{high}) \wedge \geq((x,\text{allowance}),15)$

As it was previously explained, the first step is the instantiation of the variables and the propagation of the instances to all the conditions in the constraint in which those variables appear.

$\exists a_{CUSTOMER} \quad \forall b_{OPERATION}$

$= (((a, \textit{client-assessment}),\textit{interesting}) \wedge \geq ((b,\textit{period}),8)) \vee$

$> ((b,\textit{volume}),100)) \wedge$

$(\neg = ((b, \textit{factor-recommendable}),\textit{high}) \vee$

$\neg \geq ((a,\textit{allowance}),15))$

In order to obtain the context associated with the constraint, each of the instantiated factors in the condition part must be analyzed (factor1 and factor2 from now on), and each of the instantiated conditions in the body part (condition1 and condition2 from now on). Note that both conditions have been negated. Since the constraint is positive, it will be violated whenever its condition part is satisfied and its body is not satisfied.

The context associated with the IC will contain four pairs of the type (environment, list of deductive paths), and it will be obtained by performing the operations shown in Figure 3 on the contexts associated to all the constraint components:

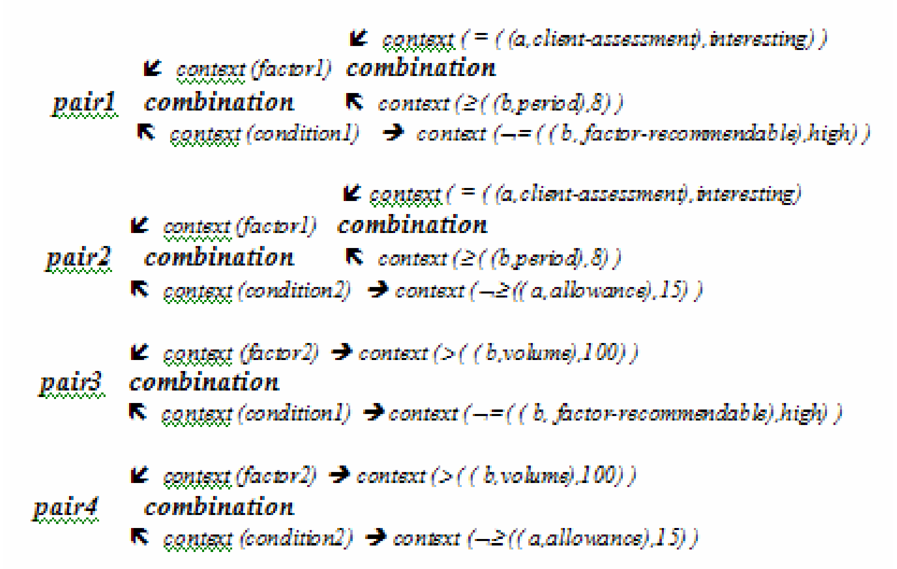
context (= ((a,client-assessment), interesting))
context (factor1)
combination
pair1
combination
context (≥ ((b,period),8))
context (condition1)
context (¬= ((b, factor-recommendable),high))
context (= ((a,client-assessment), interesting)
context (factor1)
combination
pair2
combination
context (≥ ((b,period),8))
context (condition2)
context (¬≥ ((a,allowance),15))
context (factor2)
context (> ((b,volume),100))
pair3
combination
context (condition1)
context (¬= ((b, factor-recommendable),high))
context (factor2)
context (> ((b,volume),100))
pair4
combination
context (condition2)
context (¬≥ ((a,allowance),15))

Fig. 3. Context associated with example IC

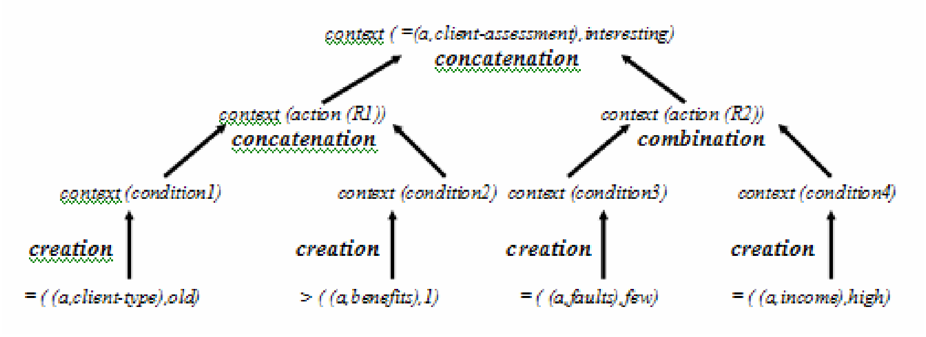

Fig. 4. Context associated with the example deducible fact

As another example, we will show how to obtain the context associated with the deducible fact:

= ((a,client-assessment),interesting)

given a rule base RB= {R1,R2} such that:

R1: = ((x,client-type),old) $\vee$ > ((x,benefits),1) $\rightarrow$
assign ((x,client-assessment),interesting)
R2: = ((x,faults),few) $\wedge$ = ((x,income),high) $\rightarrow$
assign ((x,client-assessment),interesting)

The target context will be obtained as a result of the context propagation process associated with the backward chaining of the rules in RB shown in Figure 4.

6 Conclusions

The need of validation tools for KBSs and, in special, of consistency checking tools (understanding consistency in a broad sense), will grow in the future when more critical systems are developed to be applied in industry, science, business, government, etc. The method that has been described in this paper addresses that problem, improving some of the features of previous proposals for this kind of systems.

The main advantages of CRIB over past consistency checking tools can be summarized as follows:

- Generality: CRIB can be applied on KBSs that have been developed using different representation formalisms and underlying logics. The KB-REDUCER and TADIKS tools can only be applied to KBSs with propositional logic.
- Flexibility and expressive power of its constraint specification language, that allows CRIB to detect a wide range of inconsistencies in a KB, in contrast to KB-REDUCER and TADIKS that only detect semantic and logic incompatibilities.
- The detection of potential inconsistencies covers all the possible valid inputs to the KBS without forcing the user to explicitly generate all of them and to execute the tool once for each one, as KB-REDUCER and TADIKS do.
- Incrementality: CRIB allows, in contrast to their predecessor tools, the reuse of previously derived results, when the set of integrity constraints imposed over a KB is incremented.

We can conclude that the problem of inconsistency detection in KBSs is still far from being a solved problem, despite the large number of methods and tools that have been developed during the last fifteen years. No one is general enough to be applicable to any kind of KBS. Advances in Knowledge representation and inference mechanisms leave the V&V field behind. The CRIB method is just an attempt to stay aware of and deal with all the complexity that today´s KBSs can encompass.

7 Future Research

CRIB opens important roads for future works:

- Improvement of the integrity constraint declaration process by means of an interactive editor.
- Adaptation of the method to accept KBSs represented in CCR-2, a new common conceptual representation language that improves multiple characteristics of CCR [42]. This will allow the implementation of CRIB in the revised VALID environment.
- The adaptation of CRIB to accept KBs represented in CCR-2 implies a complete redesign of CRIB since CCR-2 is a KB formalism far more powerful than CCR. As an example, this new CRIB's design should allow us to include arithmetic expressions in the rules.

7.1 Adaptation of CRIB to CCR-2

The most important innovations (related to tools like CRIB) introduced by CCR-2 with respect to CCR are:

- *CCR-2 supports the representation of a higher number of object types in the FBs:* frame classes and instances, relations, propositions, attribute values, and attribute identifiers. New predicates and actions are introduced in order to use these new types of objects. Besides, CCR-2 provides us with the possibility to represent variable declarations for all these types of objects while in CCR and, consequently, in CRIB´s IC specification language, variables can only be of the types "instance" or "attribute value".

- *In CCR-2, as well as in CCR, it is possible to represent actions to create or destroy objects while executing the KBS*. These dynamically created objects, however, in CCR cannot be bound to variables, so it is impossible to define ICs on them.
- *In CCR-2 it is also feasible to bind variables with objects created by rules*. This CCR-2 characteristic allows us to represent some types of non-monotonic reasoning.
- *In CCR-2* it is possible to include certainty factors in the rules, therefore *uncertain reasoning may be represented*.

In order to deal with these new possibilities, some modifications have to be introduced into CRIB's design, such as:

a) *A new context structure*:

- The environment will be composed of a set of objects' patterns instead of a set of predicates and metapredicates. Each pattern would show all the features that one object should have in the initial FB. These features will be expressed by constraints. For example, if an object is a frame instance of CAR then some constraints may be: "color is white", "the car is a sport car" (SPORT CAR being a category in the cars hierarchy), "John owns the car" ("own" being a binary relation), etc.
- A rule could be executed once or more times consecutively, therefore it will be necessary to represent this fact in the patterns as well as in the deductive paths in the new CRIB's design. Some of the conditions over the objects will have to be expressed as a function of the number of times that the rule is executed.
- Because of the changes in the context structure, it will be necessary to redefine the context operations: creation, combination and concatenation.

b) *An extension of the IC grammar*: since there are new types of objects in CCR-2, the IC language will have to allow us to include conditions over these objects in the ICs. Hence, the new grammar will permit the declaration of variables of whatever type of object that could appear in a CCR-2 FB. In addition, it will be possible to specify quantifications for those variables.

c) *The new process to obtain the context associated to an IC* should deal with the following problems:

1. Some types of non-monotonic reasoning.
2. Chaining between rule actions and premises: this question will be more complex than in CRIB since in CCR-2 there are predicates and actions over relations, and it accepts the presence of arithmetic expressions in the rules.
3. The creation and modification of constraints to be included in the object patterns.
4. Dealing with arithmetic expressions like constraints. These expressions may contain references to certainty factors and attribute values.
5. The detection of inconsistencies during the process: if the process to obtain the context associated to a goal builds an invalid deductive path, the contradiction should be detected in a subsequent process step.

References

1. **Plaza, E.:** KBS Validation: From Tools to Methodology. IEEE Expert, vol. 8, No. 3, 1993, pp. 45-47.
2. **Hoppe, T., and Meseguer, P.:** VVT Terminology: A proposal. IEEE Expert, Vol. 8, No. 3, 1993, pp. 48-55.
3. **Laurent, J.:** Proposals for a valid terminology in KBS Validation. Proceedings ECAI 92, 1992, pp. 829-834.
4. **Adrion, W.:** Validation, verification, and testing of computer software. ACM Computer Surveys, vol. 14, No. 2, 1982, pp. 159-192.
5. **ANSI/IEEE Standard 729, Standard Glossary of Software Engineering Terminology:** 1983.
6. **Suen Ch. Y., Grogono P.D., and Shinghal. R.:** Verifying, Validating, and Measuring the performance of Expert Systems. Expert Systems With Applications, Vol. 1, No. 2, 1990, pp. 93-102.
7. **Green, C., and Keyes, M.:** Verification and Validation of expert systems. Proceedings Western Conference on Expert Systems, 1987, pp. 38-43.
8. **Rousset, M.:** On the Consistency of Knowledge Bases: The COVADIS System. Proceedings ECAI 88, 1988, pp. 79-84.
9. **Ginsberg, A.:** Knowledge-Base Reduction: A New Approach to checking Knowledge Bases for Inconsistency and Redundancy. Proceedings AAAI-88, 1988, pp. 585-589.
10. **Beauvieux, A., and Dague, P.:** A General Consistency (Checking and Restoring) Engine for Knowledge Bases. Proceedings ECAI 90, 1990, pp. 77-82.
11. **Loiseau, S.:** Refinement of Knowledge Bases Based on Consistency. Proceedings ECAI 92, 1992, pp. 845-849.

12. **Meseguer, P.:** Incremental Verification of Rule-Based Expert Systems. Proceedings ECAI 92, 1992, pp. 840-844.
13. **Van Melle, W., Shortliffe, H., and Buchanan:** G. , EMYCIN: A Knowledge Engineer´s Tool for Constructing Rule-Based Systems. Rule-Based Expert Systems, Addison-Wesley, 1984, pp. 301-313.
14. **Suwa, M., Scott, A.C., and Shortliffe, E.H.:** An approach to verifying completeness and consistency in a rule based expert system. AI Magazine, fall 82, 1982, pp. 16-21.
15. **Steinmetz R. and Theissen S.:** Integration of Petri Nets into a Tool for Consistency Checking of Expert Systems with Rule-Based Knowledge Representation. Proceedings 6th. Workshop on Petri nets, 1985, pp. 35-52.
16. **Murata T., and Matsuyama K.:** Inconsistency Check of a Set of Clauses Using Petri Net Reductions: EECS #86-12 Dept. University of Illinois at Chicago, Technical Report 1986.
17. **Meseguer, P.:** A New Method to checking Rule Bases for Inconsistency: A Petri Net Approach. Proceedings ECAI 90, 1990, pp. 437-442.
18. **He X., Chu W. C., Yang H., and Yang S. J. H.:** A New Approach to Verify Rule-Based Systems using Petri Nets. Proceedings of the Twenty-Third Annual International Computer Software and Applications Conference (Cat. No.99CB37032). IEEE Comput. Soc, Los Alamitos, CA, USA; xx+478: 1999, pp. 462-467.
19. **Nguyen, T.A., Perkins, W.A., and Pecora, D.:** Knowledge Base Verification. AI Magazine, vol. 8, No. 2, 1987, pp. 69-75.
20. **Nguyen T.A.:** Verifying Consistency of Production Systems. Proceedings of the 3rd. conference on Artificial Intelligence Aplications, 1987, pp. 4-8.
21. **Nazareth D.L.:** An Analysis of Techniques for Verification of Lodical Correctness in Rule Based Systems, Ph D. diss., Departament of Managerial Studies, Case Western Reserve University, Cleveland, Ohio, 1988.
22. **Nazareth D.L.,** Kennedy M.H.: Static and dynamic verification, Validation and testing: The Evolution of a Discipline. Procedings of the AAA´90 on Knowledge-Based Systems Validation, Verification and Testing, Boston, 1990.
23. **Barr V.B.:** Incorporating Uncertainty in a DAG-Based Approach to Static and Dynamic Verification of Rule-Based Systems. Proceedings of the AAAI´93 Workshop on Knowledge-Based Systems Validation, Verification and testing, 1993, pp. 129-130.
24. **Ramaswamy M., Sarkar S., and Chen Ye Sho:** Using directed hypergraphs to verify rule-based expert systems. IEEE Transactions on Knowledge and Data Engineering, vol. 9, No. 2, 1997, pp. 221-237.
25. **Gursaran G. S., Kanungo S., and Sinha A. K.:** Rule-base content verification using a digraph-based modelling approach. Artificial Intelligence in Engineering, vol 13, No. 3, pp. 321-336.
26. **de Antonio, A.:** Sistema para la verificación estructural y detección de inconsistencias en Bases de Conocimientos, Trabajo Fin de Carrera, FIM, UPM, 1990.
27. **Dahl M., and Williamson K.:** A verification Strategy for Long-term Maintenance of Large Rule-Based Systems. Workshop Notes of the AAAI´92 WorkShop on Verification and Validation of expert Systems, 1992.
28. **Ayel M.:** Protocols for Consistency Checking in Expert System Knowledge Bases. Proceedings of the th. European Conference on Artificial Intelligence (ECAI'88) 1988, pp. 220-225.
29. **Ayel M.,** and Laurent J. P.: SACCO-SYCOJET: Two Different Ways of Verifying Knowledged-Based Systems. Validation, Verification and Test of Knowledge-Based Systems: John Wiley Publishers, 1991, pp. 63-76.
30. **de Kleer J.:** An Assumption Based TMS. Artificial Intelligence vol. 28, No. 2, 1986, pp. 127-162.
31. de Antonio, A.: Una interpretación Algebraica de la Verificación de Sistemas basados en el Conocimiento, Ph.D. diss, Facultad de Informática, Universidad Politécnica de Madrid, 1994.
32. **Laita L. M., Roanes-Lozano E., Roanes-Macías E., and Díaz A.:** From Computer Algebra to AI. Application to Verification and Automated Proving in Multi Valued Logics Knowledge Systems. Proceedings of the SEKE-97, 1997, pp. 295-301.
33. **Laita L. M., Roanes Lozano E., de Ledesma L., and Maojo V.:** Computer Algebra based Verification and knowledge extraction in RBS application to Medical Fitness criteria. Proceedings of the EUROVAD'99, 1999.
34. **Antoniou, G.:** Verification and Correctness Issues for Nonmonotonic Knowledge Bases. International Journal of Intelligent Systems, Vol. 12, No. 10, 1997, pp. 725-738.
35. **Wu C. H., and Lee S. J.:** Knowledge Verification with an Enhanced High-Level Petri-Net Model. IEEE Expert, Sep/Oct, 1997, pp. 73-80.
36. **O'Leary D. E.:** Verification of uncertain knowledge based systems: an empirical verification approach. Management Science. Vol. 42, No. 12, 1996, pp. 1663-1675.
37. **Lee S., and O'Keefe R. M.:** Subsumption anomalies in hybrid knowledge based systems. International Journal of Expert Systems, Vol. 6, No. 3, 1993, pp. 299-320.
38. **Mukherjee R., Gamble R. F., and Parkinson J. A.:** Classifying and detecting anomalies in hybrid knowledge-based systems. Decision-Support-Systems, Vol. 21, No. 4, 1997, pp. 231-251.
39. **Cardeñosa, J., and Juristo, N.:** General Overview of the Valid Project. Proceedings European Symposium on the Validation and Verification of Knowledge Based Systems, EUROVAV'93, 1993, pp. 53-67.
40. **VALID team, Deliverable D4:** VETA definition, in ESPRIT II 2148 VALID project, 1989.
41. **VALID team, Deliverable D10:** Report on Valid Environment. ESPRIT II 2148 VALID project, 1990.
42. **Martínez, L.:** CCR2: Un Modelo Genérico de Representación del Conocimiento. Trabajo Fin de Carrera, FIM, UPM, 1993.

Julia Clemente. *She obtained in 1994 the Computer Science degree from the Technical University of Madrid (Spain). She is currently assistant professor in the Department of Automatica at the University of Alcala. Her areas of research are the Verification of Knowledge Base Systems and Intelligent Viirtual Training Environments. Nowadays she is working for his Ph.D. in the area of Student Modelling, applied to Intelligent Tutoring Systems and Virtual Environments.*

Angélica de Antonio. *She is associate professor at the Computer Science School of the Technical University of Madrid, Spain (UPM), where she has taught subjects related to software engineering, project management and databases since 1990. She is also Director of the "Decoroso Crespo Laboratory for Educational Applications of Computing" in the UPM since 1994, where she has lead several research and development national and international projects. Her research interests include intelligent agents, artificial intelligence, virtual environments, educational software and software engineering. She received a B.S. in 1990 and a Ph.D. in Computer Science in 1994 from the Technical University of Madrid.*

Jaime Ramírez. *He obtained his Bachelor degree in Computer Science by the Technical University of Madrid in 1996 and his Ph. D. in Artificial Intelligence by the Artificial Intelligence department of the Computer Science School (Technical University of Madrid) in 2002. His research activities mainly have been related to the field of Verification of Knowledge Base Systems, and the development of applications based on Virtual Reality and Intelligent Learning. Currently, he teaches Computer Programming courses in the Technical University of Madrid, and he works in the Decoroso Crespo Laboratory (Technical University of Madrid), as a collaborator professor.*