An approach for automating the verification of KADS-based expert systems

Abeer El-Korany
Central Lab. For Agricultural Expert Systems (CLAES), P.O. Box: 100 Dokki, Giza, Egypt
abeer@esic.claes.sci.eg

Khaled Shaalan
Computer and Information Science Dept., Institute of Statistical Studies and Research (ISSR),
Cairo Univ., 5 Tharwat St., Orman, Giza, Egypt
shaalan@esic.claes.sci.eg

Hoda Baraka
Hbarka@idsc1.gov.eg
Computer Engineering Dept., Faculty of Engineering, Cairo University, Dokki, Giza, Egypt

Ahmed Rafea
Computer and Information Science Dept., Institute of Statistical Studies and Research (ISSR),
Cairo Univ., 5 Tharwat St., Orman, Giza, Egypt
rafea@esic.claes.sci.eg

Abstract cation plays an important role in assuring the reliability of expert systems. Expert systems verification involves checking the knowledge base for consistency, completeness, and other errors. Our study indicates that in order to verify an expert system, it is necessary to have a *conceptual model* of the knowledge base. The KADS methodology lends itself to conceptual modeling of the knowledge base. This enabled us to build an automatic verification tool. This tool is able to detect different knowledge base error types. A novel feature of this tool is its ability to detect consistency errors that arise due to using KADS methodology in knowledge modeling.

1. Introduction

The importance of assuring the quality of expert systems is now widely recognized. Quality assurance is a major issue in development of expert systems. A consensus has been reached in the literature that the evaluation of expert systems to ensure their reliability involves two principle activities, usually called verification and validation (V&V). Studies have shown that *verification*, can lead to the early detection of errors that otherwise would have remained even after extensive validation tests (1,2,3). Verification, therefore

is an important part of reliability assurance for these systems, and it is the interest of all expert system builders to ensure that verification is performed on their system prior to traditional methods of testing. Our proposed approach for verifying knowledge bases is based on using conceptual modeling for knowledge base (KB). Conceptual models, such as supported by KADS methodology (4,5) make it possible to derive the structure of knowledge base systems (KBS). This enabled us to build an automatic verification tool. This tool is able to detect different KB error types such as consistency and completeness errors, as well as other errors. A novel feature of this tool is its ability to detect consistency errors that arise due to using KADS methodology in knowledge modeling.

The structure of the paper is as follows. Section 2 discusses the needs for assuring the quality of expert systems. Section 3 presents the different approach in verification and validation of expert systems. Section 4 briefly describes the knowledge base development environment upon which we build our verification tool. Section 5 examines the proposed verification tool in details. Section 6 studies the ability of verification tool in verifying some real-world knowledge bases. Section 7 summarizes the paper and considers some open research issues.

2. The needs for assuring the quality of expert systems

Several factors have led to need for assurances about the quality of expert systems (6). One such factor is a new generation of expert systems which are beginning to appear. These expert systems are embedded within another hardware/software system in such a way that users may not even realize that an expert system is a part of the software.

Another factor is that expert systems are being proposed for mission-critical application. In such applications, there is a possibility of great financial loss and/or loss of life if the expert system malfunctions.

An additional factor is the expectations and standards for quality software which the commercial computing community has already established. The

commercial computing community is unlikely to lower its expectations or standards of quality in order to adapt to expert systems. Expert system developers will have to adapt to the preestablished expectations and standards of commercial computing community.

3. Different approaches in verification and validation of knowledge- based systems

During the first years of KBS production, researchers thought of KBS life cycle as rapid prototype, while the V&V was handled on as-needed basis. Recently, methods (and their support tools) were developed to detect problems in rule-based systems, such as redundant, subsumed, or missing rules. However the field lacked a comprehensive view of KBS validation and the role of V&V in the KBS life cycle (7). Two approaches have been realized for them.

One approach discusses the relation of KBS V&V to the *software engineering methodology* for program correctness and evaluation. This discussion dealt with the usability and usefulness of formal program specification for expert systems. Traditional methods of software testing produce empirical measures of reliability: they involve running test cases through the system and evaluating the correctness of the result obtained.

A second approach is the use of *conceptual modeling frameworks* such as KADS (8) and the Components of Expertise (9) as a new kind of specification for KBS design. Knowledge models provide a framework in which include the V&V of pragmatic constrains, and the progressive refinement of knowledge models provides principled mapping to the symbol level necessary for V&V (2).

4. A brief description of the knowledge base development environment

The major challenge for any modeling approach to KBS construction is to find an adequate answer to the question of how to model expertise. The requirement of the resulting model of expertise, that is of a knowledge-level nature: independent of a particular implementation (10). KADS is a method for the

structured and systematic development of KBS, which aims to provide software engineering support for the knowledge engineering process (11,12).

Other observed difficulties in building KBS result from the kind of language used to build these systems, since the conceptual model must have an explicit semantic foundation. KROL (13,14) has been successfully used in building several KADS-based expert systems at CLAES (Central Laboratory of Agriculture Expert System) at the Agriculture Research Center of Ministry of Agriculture and Land Reclamation in Egypt such as: CITEX (15) and CUPTEX (16).These expert systems are in actual use by the Egyptian extensions service. The following sections give an overview for the knowledge base development environment upon which we built our verification tool. However, it is not intended to constitute an introduction to this topic, and suitable reading material will be referred to during discussion.

4.1 KADS: A methodology for modeling knowledge-based systems

KADS (17) is a methodology that has been developed in the framework of the Esprit program. The model-based approach according to KADS is rapidly becoming the de facto standard in Europe.

In (18), we described our approach for building KADS-based expert systems. Briefly, the *domain knowledge* identifies the domain in terms of concepts, properties of these concepts, and relations. Fig 1. Shows an example of the relation between expressions *soil_determine_soil*, which means that the soil parameters are to be determined by using other soil parameters. The *inference knowledge* is a declaration of what primitive inferences can be made using the domain knowledge. The *task knowledge* describes the steps that must be carried out in order to achieve a particular goal using the inference steps of the inference layer.

LEFT HAND SIDE			RIGHT HAND SIDE		
Concept	Property	Value	Concept	Property	Value
Soil	Texture	salt loam	Soil	S_Status	suitable
	Profile Depth	>= 1.2			
	ESP	=< 15			
	Ph	=< 8			
Soil	Texture	loam	Soil	Type	medium
		sily loam			
		sandy clay loam			
Soil	Texture	sand	Soil	Type	coarse
		sandy loam			

Figure 1: An Example of a relation between expressions

4.2 KROL: The language

Our implementation language KROL, (13) is a Knowledge Representation Object Language that addresses the representation of knowledge. KROL is built on top of SICStus Prolog language (19).

The applicability of KROL as a language to implement KADS-based expert systems could be described as follows. Domain knowledge is represented by a single formalism, the object. Objects correspond to real-world concepts or rules. Rules are uniformly handled in an object-oriented manner. The behavior of an object is represented by methods and its characteristics are represented by properties (attributes). Attributes may have facets (value type, value source, possible values, and if the property takes a single or multiple value). The relation between concepts can be represented by the method *super* within the concept. A particular relation between expressions is manifested as a set of declarative rule instances defined in an object. A rule instance, or simply a rule, is declared by writing it in the following form:

ruleid(conclusion) ***if*** *premise*

The inference knowledge is represented by an object with inference steps as its behavior. Each inference step either uses the primitive inference defined by the built-in *inference_class* object, e.g. *conclude_all* primitive which tries to prove all rules in a given relation between expressions and recursively its descendents.

The task knowledge is represented by an object that its behavior describes the application of inference steps that satisfies a particular goal using the inference steps of the inference knowledge. Fig.2 illustrates a sample of each KADS layers using KROL.

Soil_determine_soil :: { super(assessment_system) & r1([s_status of soil = suitable]) if texture :: soil = 'salt loam' profile_depth :: soil >= 1.2, esp :: soil =< 15, ph :: soil =< 8.0, calcium_c_c :: soil =< 40.0 & r2([s_status of soil = unsuitable]) if texture :: soil = 'loam' -> true ; profile_depth :: soil < 1.2 -> true ; esp :: soil >= 15 -> true ; ph :: soil >= 8.5 -> true ; calcium_c_c :: soil > 40.0	assesment_inference :: { super(inference_class) & abstract:- inference_class:: conclude_all(soil_determine_soil), inference_class::conclude_all(water_determine_water), ... assign:- inference_class::conclude_all(soil_determin_conclusion), inference_class::conclude_all(water_determin_conclusi), ... }.	assessment_task :: { ... start:- assesment_inference::abstract, assesment_inference:: assign }.
Relation between expressions	Inference layer	Task layer

Figure 2: An example of the KADS layers implemented using KROL

5. The methodology

Verification examines the technical aspects of an expert system in order to determine whether the expert system is built correctly. One of the major tasks in verifying expert systems is the verification of the knowledge contained within the KB. Verifying the KB involves examining the consistency, completeness, and correctness of the knowledge by detecting errors such as redundancy, contradiction, and circularity (6). Several verification criteria have been proposed. Our approach in automating verification of the KB involves checking the KB for commonly occurring errors. The approach indicates also that, in order to verify an expert system, it is necessary to have a *conceptual model* of the KB. The KADS methodology lends itself to conceptual modeling of the KB. This enabled us to build an automatic verification tool. This tool is able to detect different KB error types as well as new error types that appear due to using KADS in knowledge modeling.

A major design goal of our verification tool is to be generally applicable for any KADS-based expert system. KROL is successfully used to implement several KADS-based expert systems at CLAES. On the other hand, this implementation faces some limitations. One of these limitations, is that there is not a *complete transformation mapping* between the conceptual model and the implementation language. Furthermore, the design of our verification tool depends mainly on the conceptual modeling of the KB. Thus, KROL is extended to support the complete transformation mapping by defining a set of KROL methods. These methods affect the three levels of knowledge layers. Representation of each knowledge layer after adding the KROL extensions, highlighted in bold, is described in Fig.3. The following sections, describe our approach for automating the verification of KADS-based expert systems.

<table>
<tr>
<td>Soil_determine_soil :: {

super(assessment_system) &

input_att(soil,texture/1)&
input_att(soil,profile_depth/1)&
input_att(soil,esp/1)&
input_att(soil,ph/1)&
input_att(soil,calcium_c_c/1)&
output_att(soil,s_status/1)&

r1([s_status of soil = suitable]) if
texture :: soil = 'salt loam'
profile_depth :: soil >= 1.2,
esp :: soil =< 15,
ph :: soil =< 8.0,
calcium_c_c :: soil =< 40.0 &

r2([s_status of soil = unsuitable]) if
texture :: soil = 'heavy clay' -> true
; profile_depth :: soil < 1.2 -> true
; esp :: soil >= 15 -> true
; ph :: soil >= 8.5 -> true
;calcium_c_c :: soil > 40.0
}.</td>
<td>Abstract:: {

super(assessment_system) &

input-role([system_description],[[soil,texture],...]) &

output-role(abstracted_system_description,
[[soil,s_status],...]) &

static-role([soil_determine_soil,...]) &

. . .

}.</td>
<td>assessment_task :: {

super(krol) &

inference([abstract,assign]) &

start:-
assesment_inference :: abstract,
assesment_inference :: assign

}.</td>
</tr>
<tr>
<td>Relation between expressions</td>
<td>Inference layer</td>
<td>Task layer</td>
</tr>
</table>

Figure 3: Extending KADS layers using KROL

5.1 Structure of the verification tool

The verification process of KADS-based expert systems can be distinguished into three main parts:

1. ***Domain knowledge verification.*** During this process, we are focusing on the domain knowledge which contains concepts, properties, relation between

concepts, and relation between expressions. Most of the KB errors will be detected in this part.

2. ***Inference Layer Verification.*** In KADS, an inference layer inconsistency may occur. This happens when an input/output role of any inference step has a defined input/output data elements that are not defined in the corresponding relation between expressions of the domain layer. Moreover, when an inference has a defined input-role that is not produced as output-role of another inference step.

3. ***KADS Layers verification.*** When applying KADS methodology in knowledge modeling, new types of error are discovered. The three layers that construct the knowledge model are interrelated, since each layer always refers to some parts of another layer. According to this interaction, inconsistencies between layers may arise. Fig.4 gives an overview of the structure of our proposed verification tool.

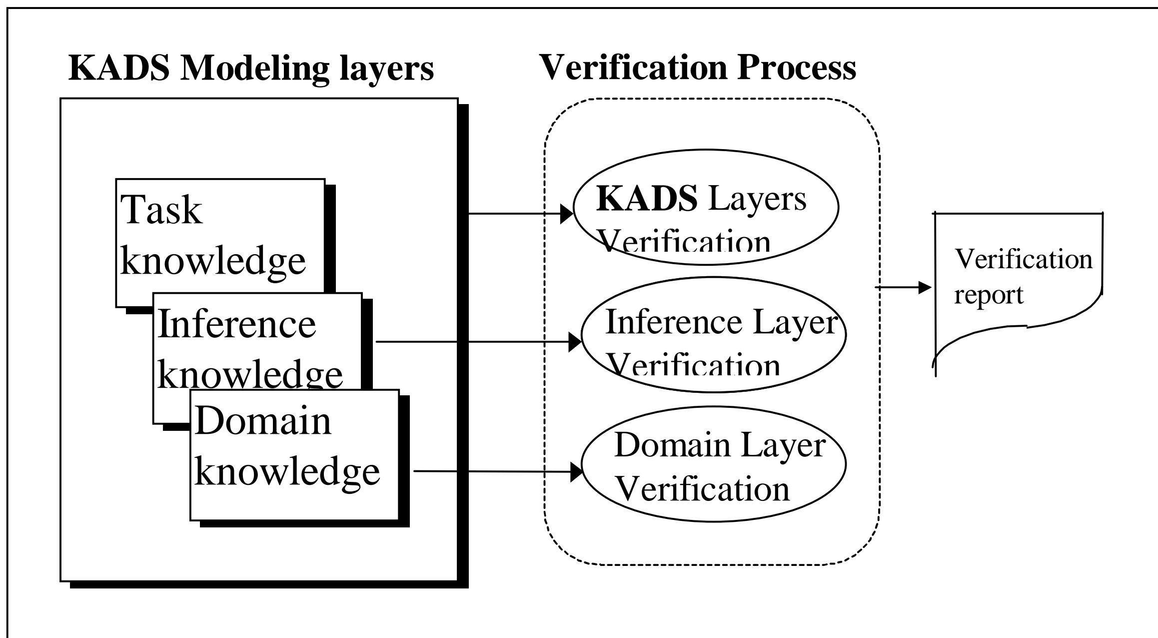


Figure 4: Overall structure of the verification tool

5.2. Domain knowledge verification

The domain knowledge verification process detects most of the coded KB errors. The verification process considered here is divided into three phases, according to the type of errors detected in each phase. They are:

1. Consistency checker phase.

2. Check for completeness phase.

3. Path checker phase.

Fig. 5 illustrates the structure of the domain knowledge verification process.

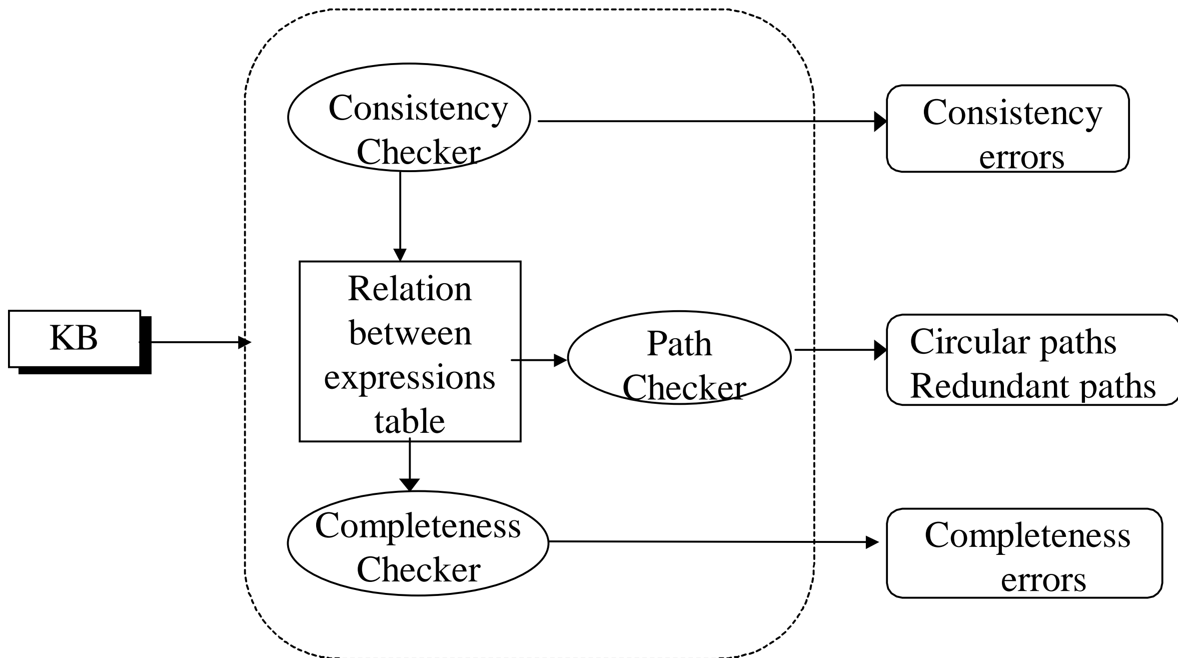


Figure 5: The structure of verification of the domain knowledge

5.2.1 Consistency checker phase

The *consistency checker* works on the relations between expressions of the domain layer, one relation at a time. Consistency of the KB appears as: undefined object, undefined attribute, undefined attribute values, duplicate rule pairs, conflict rule pairs, and subsumed rule pairs. Errors in spelling or syntax are frequent source of consistency errors. The main function of the consistency checker is *detecting consistency error.* A secondary function is creating *the relation between expressions table* to support the subsequent verification processes.

Detecting consistency errors. For each relation between expressions, the consistency checker checks each rule instance to find out the undefined object,

undefined attribute, and undefined attribute values. This is realized by comparing the objects, attributes, and attribute values used in each rule against their corresponding definitions. For example, consider the attribute *soil status* with the set *{sandy, heavy, light}* as its legal values. This attribute has a defined source of value to be *derived*[1]. If a rule refers to the value *sand* for the attribute *soil status*, it is detected as undefined attribute value.

Detecting duplicate, conflict, and subsumed rule pairs are realized by comparing each rule against every other rule within the same relation between expressions.

Creating the relation between expressions table. The relation between expressions table contains the needed information about all the relations between expressions in the KB. The basic idea behind constructing this table is to accelerate searching for any defined attribute in the KB which is heavily used in subsequent phases (See Fig. 5). This table consists of the following fields:

- **Relation name**: The name of the relation between expressions as defined in the KB.
- **Input attribute**: The names of object-attribute pairs given in the rules antecedence.
- **Output attribute**: The names of object-attribute pairs given in the rules consequence.

5.2.2 Completeness checker phase

As the number of rules grows large, it becomes impossible to check every possible path through the system (20).There are four indicative situations of gaps in the knowledge base: unused attribute values, missing rules, unfirable rules, and unused consequence.

---

[1] KROL distinguishes between three types of value source: user when the attribute value is input by the user, database when the value is queried from a database, derived when the value is concluded by a rule.

The purpose of *check for completeness phase* is to scan the whole KB looking for unused attribute values, missing rules, unfirable rules, and unused consequence. For efficiency reasons, detecting such errors is divided into two main parts: *detecting unrefrenced attribute value and missing rules*, and *detecting unfirable rules and unused consequence.*

Detecting unrefrenced attribute value and missing rules. The unrefrenced attribute values are detected when we do not find any of the attribute values in any rule antecedence. While missing rules are detected when the unused values are derived from a rule consequence. For example, consider again the attribute *soil status*. If both the value *light* and *heavy* are only used in rules antecedence part, then the value *sandy* is detected as an unused attribute value. On the other hand, if this is the case with rules consequence part, then there is a rule missing for the value *sandy*.

In order to detect unrefrenced attribute values and missing rules, it is necessary for each defined attribute to get all its given values through the relation between expressions and compare it with its defined legal values. Using the relation between expressions table, we are able to extract all used attribute values in each relation and detect the unused attribute values.

Detecting unfirable rules and unused consequence. The unfirable rule is detected when one of the given attributes in the rule antecedence part has a defined source of value to be derived and the attribute does not appear in any rule consequence part. This means that the attribute value that would have determined by the missed rule would never fire. For example, consider the derived values of the aforementioned attribute *soil status* that is used in rules antecedence part of the relation between expressions *soil_determine_soil*. Moreover, these attribute values are not derived from any other relations of the system. In this setting, rules of this relation will never fire.

On the other hand, if the rule consequence is neither one of the final goals, nor it appears in any rules antecedence then it is unused consequence.

5.2.3 Path checker phase

The last phase of the domain knowledge verification process concerns detecting circular and redundant paths. These paths will be detected from a graph data structure. This graph links the input attributes to the output attributes for each defined relation between expressions using the relation between expressions table. Detecting these errors is divided into two main parts: *detecting redundant paths*, and *detecting circular paths*. The following describes how to traverse this graph to detect each of these errors.

Detecting redundant paths. A redundant path is found when it is possible to reach the same conclusion from the same inputs through different paths. For example, consider the attribute *material qty* of the irrigation subsystem which could be reached through two different paths that originate from the same objects. The following notational conventions are used: an arrow indicate an arc, a comma separates concepts, and a colon separates an object/attribute pair. The first path is obtained from rules of the relation:

plantation,irrigation_system→ [irr_op:material_qty]

Whereas, the second path is obtained from rules of more than one relation:

plantation,irrigation_system→ [plant:ad]
→ [irrigation:I]
→ [irrigation:lr]
→ [irrigation:wr]
→ [irr_op:material_qty]

This process is repeated to obtain all possible paths that connect each output attribute to other attributes for all the relations between expressions. If edges of any two paths are identical, a redundant path is reported.

Detecting circular paths. Circular paths are detected when an attribute appears as an input attribute of one relation and as an output attribute of another relation and a path between the other edges of these relations can be reached. For example, if we have the following two paths:

path1: *[plant:rd]→ [plant:rdf]*

path2: *[plant:rdf]* $\rightarrow$ *plantation,irrigation_system* $\rightarrow$ *[plant:rd]*

A circular path is reported because it is possible to reach the attribute *rd* of the *plant* concept from the same input through following *path1* then *path2*.

5.3 Inference Layer Verification

The verification process considered here is divided into two phases:

1. Step checker phase.

2. Inference checker phase.

Fig. 6 illustrates the structure of the inference knowledge verification

5.3.1 Step checker phase

The *step checker* works on the inference steps of the inference layer. The main functions of the step checker are detecting *inference step consistency error,* and *creating the inference table*.

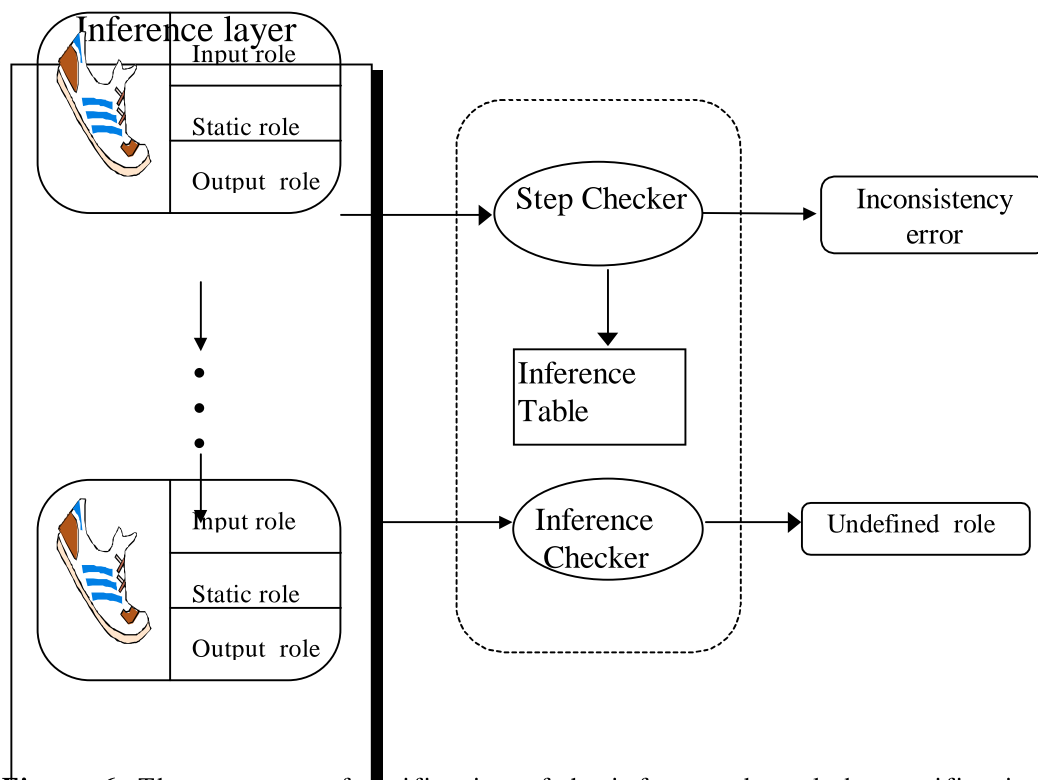


Figure 6: The structure of verification of the inference knowledge verification process.

Inference step consistency errors. The inference layer consists of inference steps. Each inference step operates over data elements corresponding to the domain layer. The input-role refers to a list of input data elements of the inference step. These elements correspond to a combination of the input-attributes of the relation between expressions which this inference uses. Also, the output-role refers to a list of output data elements of the inference step. These elements correspond to a combination of the output-attributes of the relation between expressions which this inference uses. Inconsistency arises when the input- or output-role refers to data element that is not defined in any relation between expressions of this inference. This is best clarified by an example. Consider the input-role: *system_description* of the inference step *specify* shown in Fig.7 that has the following defined input data:

[soil:texture], [soil:ec], [soil:ph], and *[water:boron]*

This role works on following relations between expressions:

soil_determine_soil and *water_determine_water.*

These relations have the following input-attributes:

[soil:texture], [soil:ec], [soil:ph], [water:boron], [water:sar], and *[water:rsc].*

Obviously, inconsistency is reported because the input attributes *sar* and *rsc* of the concept *water* are defined by the relations between expressions *soil_determine_soil* and *water_determine_water* while missed in the input attributes of the input role for the same inference step *specify*.

Creating the inference table. The step checker creates an inference table in order to detect KADS-based errors. This table is created in order to facilitate the detection of these inconsistency errors. The table consists of the following fields:

- **Inference name**: The name of the inference step as define in the KB.
- **Input role**: The input-role name(s) of the inference step.
- **Output role**: The output-role name of the inference step.
- **static role**: The list of the relation between expressions that are used by this inference.

5.3.2 Inference checker phase

The *inference checker* works on the input/output roles of the inference layer. Since each inference has a defined input-role and output-role, each output-role should either be an input-role to the following inference step or the last output. Fig. 7 depicts the inference layer of an agriculture expert system. As shown in this figure, the inference step *specify* has an input-role *system description* and an output-role *specified case description* which in turn is an input to the next inference step *compute*. The last inference step is the only one that its output-role is not input to any other inference steps. Thus, another type of inconsistency of the inference layer may arise when one of the inference steps has a defined output-role that does not satisfy either of the above cases. In order to detect such inconsistency the inference table is used to ensure that each defined *output-role* matches one of the defined *input-roles* for another inference step.

5.4 KADS layers Verification

The layers of the KADS modeling methodology always have a limited interaction. This interaction could introduce new types of inconsistency errors. The verification process considered here concerns consistency of the KADS layers. Fig.8 illustrates the structure of the KADS layer verification process.

Detecting layers consistency errors. Each knowledge layer of the KADS modeling methodology always refers to some parts of another layer. For example, in the task layer, tasks apply the inference steps defined in the inference layer. Each inference step uses one or more relation between expressions of the domain layer. This relationship is depicted in Fig.8

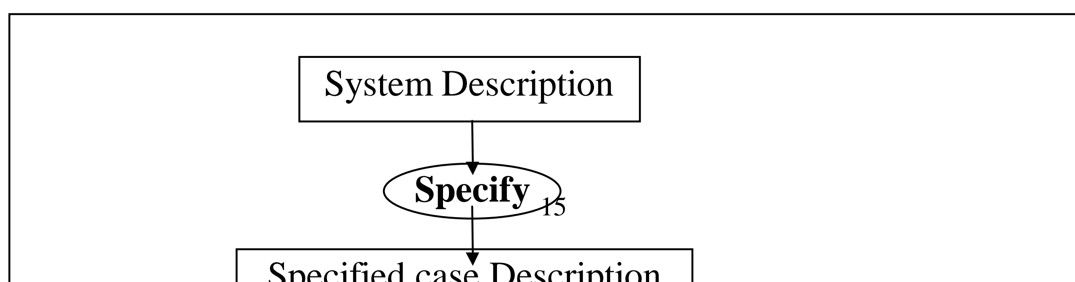
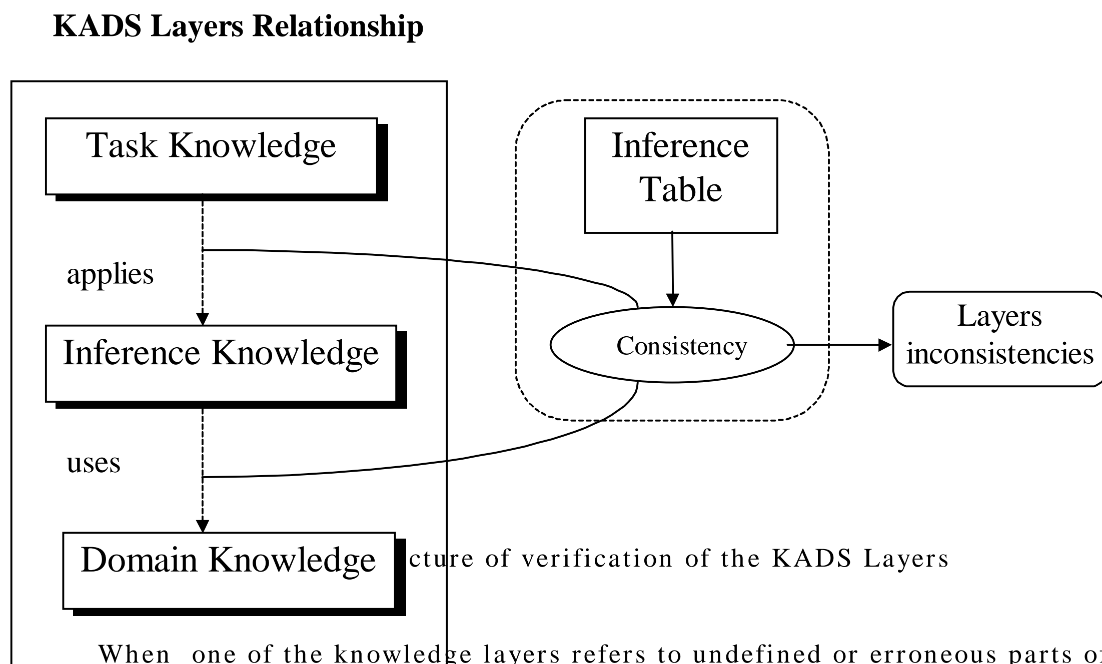


Figure 7: An inference structure for irrigation in a crop management system. Rectangles represent roles; ovals represent inference steps. Arrows are used to indicate input-output dependencies.



cture of verification of the KADS Layers

When one of the knowledge layers refers to undefined or erroneous parts of another layer, inconsistency between layers occurs. As an example, consider the

task *Asssment_t*,a task of the *assessment* task layer, that applies an inference step *specfy* which is not found in the inference layer of this subsystem. The inference would have applied the inference step *specify*. Furthermore, the inference step *abstract* uses a defined relation between expressions *s_determine_s* which also is not found in the domain layer. The domain layer would have used the *relation soil_determine_soil.*

The consistency checker uses the inference table to detect such inconsistency. For example, each inference step has its defined relation between expressions via the static role. By scanning the relation between expressions table we can determine whether these relations are already defined in the domain layer.

6. Examples of utilization and testing

This section presents the results of running our verification tool on a sample of real-world knowledge bases systems. Several examples covering a wide range of KB error types were used. These examples demonstrate the capability of our tool to discover these errors. The examples presented here were taken from an expert system for citrus crop management in open fields (CITEX) (15) which is developed by CLAES. CITEX consists of five subsystems, namely: assessment, irrigation, fertilization, diagnosis, and treatment. Each subsystem, which is considered an expert system in its own, is verified in isolation from the rest of the system and the results obtained are its local verification. As mentioned before, it was modeled using *KADS* methodology. The system contains 160 objects, 145 attributes, and 369 possible values. Moreover, it contains 632 rules and 468 factual knowledge comprising 1100 relations.

6.1 Results of Domain knowledge verification process

The results of running our tool on a sample of real-world knowledge bases show that completed knowledge bases were checked by our verification tool, that is the system was considered to have been tested satisfactorily by other means

(typically running a large number of actual test cases on the system, and evaluating the output produced).

Table 1 summarizes the results of running our verification tool on each subsystem of CITEX. It should be noted that the number of errors presented in this table are the detected errors by the verification tool. These errors are alerted to the knowledge engineer who decide whether or not they indicate an actual error. In each cell of the table there are two numbers separated by a slash. The number of errors detected by the tool is recorded to the left of the slash and the number of errors accepted by the knowledge engineer is recorded to the right of the slash.

	Assessment	**Diagnosis**	**Treatment**	**Irrigation**
Consistency errors				
Illegal values	5/5	20/20	33/33	1/1
Out of range	4/4	0	0	6/6
Undefined attribute	4/4	0	1/1	0
Conflict rules	0	1/1	0	0
Completeness errors				
Unused attribute value	6/6	43/24	29/22	1/1
Unfirable rule	0	1/0	2/0	5/0
Missing rule	3/3	1/0	0	0
unused consequence	1/0	1/0	1/0	1/0
Detect wrong paths				
redundant paths	9/0	3/0	0	6/0

Table1: Number of errors detected by the verification tool in four subsystems

From the table we can note that the number of the detected unused attribute values differs from the actual number of errors for both the diagnosis and the treatment subsystems. Due to the design of CITEX, the attributes legal values are defined in a sharable knowledge base which is common to all its subsystems. Since the tool is able to locally verify each subsystem, the detected unused attribute values in these subsystems are covered by others. This is best explained by an example. Consider the attribute *soil type* which has the following set as its legal values: *{fine, medium, coarse}*. In the diagnosis subsystem, the used values are only *fine* and *coarse*. Therefore, the tool detected the value *medium* as an unused value, while the value *medium* is used in other subsystems. Actually, this can be

treated as a warning produced by many programming languages compilers. The same situation also applies when detecting the missing rules.

detected unused consequence were final goals.

The redundant paths are detected when the output attribute could be reached from the same input attribute through different paths. The detected paths connect two object-attribute nodes. These nodes were alerted to the knowledge engineer who checked whether or not the attributes take the same values. That is why the detected errors were not considered to be actual errors.

An important point to note here is that, the actual errors detected for these real-world knowledge bases are always due to typographical errors or wrong value initialization. However, running a set of test cases is not sufficient to detect these errors. Therefore, a provision an automatic verification tool, would save much time and efforts while developing large and complex system.

6.2 Results of inference and KADS layers verification process

The result of running our tool to verify inference and KADS layers showed that only one type of inconsistency. The step checker phase found inconsistency between input-data of input-role and input attributes of the relations between expressions of this role. Regardless of this result, the goal of those verification processes were to discover errors while developing expert systems. However, the tool was applied on a developed expert system, and consequently its output has such an error. By means of contrived examples while developing our tool, we expect more errors to be reported to the developer at the developing of their expert systems.

7. Conclusions

We developed an automatic verification tool. The evolution of this tool came about because of the urgent need to verify KADS-based expert systems developed at CLAES. Our approach for designing the automatic verification tool is based on *conceptual modeling* for knowledge bases. The verification tool have been implemented on PC platform using SICStus Prolog.

We showed several examples that cover a wide range of knowledge base error types. These examples demonstrate the capability of our tool to discover these errors that could otherwise remain even after conventional testing. They were taken from CITEX, an expert system for citrus production in open field.

The authors would like to acknowledge the Central Laboratory of Agricultural Expert Systems (CLAES) at the Agriculture Research Center of Ministry of Agriculture and Land Reclamation in Egypt leaded by Prof. Dr. Ahmed Rafea for their support while conducting the research described in this paper.

REFERENCES

1. Nazareth, D.L., Issues In The Verification Of Knowledge In Rule-Based Systems, *International journal of Man Machine studies*, 30(3), pp. 255-271, 1989.

2. Plaza. E., KBS Validation: From Tools to Methodology, *IEEE*, pp. 66-77, June 1993.

3. Preece A. D, Shinghal R., Foundation And Application Of Knowledge Base Verification, *International Journal of Intelligent system*, John Wiely & Sons Inc., 9 (8), pp. 683-701, 1994.

4. Preece A. D., A New Approach To Detecting Missing Knowledge In Expert System Rule Bases, *International Journal of. Man Machine Studies*, Academic Press Limited, Vol. 38, pp. 661-688, 1993.

5. Preece A. D., *Verification Of Rule-Based Expert Systems In Wide Domain*, in N. Shadbolt (Ed.), Researcher and development in expert system VI: Proc. Expert systems 89, Cambridge University Press, pp. 66-77, 1989.

6. Smith S., Kandel A., Verification and Validation of Rule Based Expert System, CRC Press Inc., 1993.

7. Lopez B., Meseguer P., and Plaza E., Validation of knowledge based systems: A state of Art, *AI Communication*, 3 (2), pp. 56-72, 1990.

8. Wielinga B., Schreiber A., and Breuker J., KADS: A Modeling Approach To Knowledge Engineering, Knowledge acquisition, *Esprit project, Report No. 5248 KADS-II, Vol. 4, No. 1*, 1992.

9. Steels, L.,The *AI Magazine*, Vol. 11, No 3, 1990.

10. Schreiber. G, Wielinga. B, Breuker. J., *KADS: A Principle Approach To Knowledge-Based System Development*, Academic Press, London, 1993.

11. Shadbolt N., Wielinga B. J., Knowledge Based Knowledge Acquisition: The Next Generation Of Support Tools. In Wielinga, B. J., Boose, J., Gaines, B., Schrieber, G., and Someren V. (Eds), Current Trends in Knowledge Acquisition, Amsterdam, The Netherlands, 1990.

12. Voβ A., Karbach W., Implementation KADS Expertise Models with Model-K, *IEEE Expert,* pp. 74-82, 1993.

13. ESICM, *A Knowledge Representation Object Language (KROL),* Technical report, No. TR-88-024-27, 1993.

14. ESICM, *Guide To KADS Implementation Using The Knowledge Representation Object Language KROL*, Technical report, No. TR-88-024-31, 1993.

15. Salah A., Hassan H., Tawfik K., Ibrahim I., Farahat *In Proceeding of the 2nd National Expert Systems and Development Workshop (ESADW-93)*, MOLAR, Cairo, Egypt, May, pp. 63-72, 1993.

16. El-Dossouki A., Edrees S., El_Azhary S., CUPTEX: An Expert System For Crop Management Of Cucumber, *In Proceeding of the 2nd National Expert Systems and Development Workshop (ESADW-93)*, MOLAR, Cairo, Egypt, May, pp. 31-42, 1993.

17. Wielinga B., Akkermans H., Hassen H. Olsson O., Orsvan K., Schrieber G., Terpstra P., Van de Velde W., and Wells S., *Expertise Model Definition Document*, ESPRIT Project P5248, Report No. KADS-II/M2/UvA/026/5.0, University of Amsterdam, The Netherlands.

18. Rafea A., Edrees S., El-Azhari S., Mahmoud M., A Development Methodology For Agricultural Expert System Based On KADS, *In Proceeding of the 2nd world congress on Expert System*, Cognizant Communication Corporation, pp. 442-450, Jan., 1994.

19. SICStus Prolog User's Manual, Swedish Institute of Computer Science, S-164 2, KISTA, Sweden, 1995.

20. Nguyen T.A, Perkins W.A, Laffey T.J and Pecora D., Knowledge Base Verification, *AI Magazine*, 8(2), pp. 69-75, 1987.