Design of Verification Tool

For

MiniKSR

By

Dr. Abeer El_Korany

February 2003

Table of Contents

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, rule clusters, tables, and mathematical functions. Most of knowledge base 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 knowledge base components that are not defined in the domain layer. Another inconsistency error occurs 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. Since the three layers that construct the knowledge model are interrelated, each layer always refers to some parts of another layer. Accordingly, inconsistencies between layers may arise.

1 Domain knowledge verification

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

1. Syntax checker phase
2. Consistency checker phase.
3. Check for completeness phase.
4. Path checker phase.

1.1 Syntax checker phase

For each of the domain knowledge type (rule clusters, tables, and functions), we have to detect the syntax error due to typographical mistakes. A parser works on different domain knowledge components according to its internal representation to ensure that it is written correctly.

1.2 Consistency checker phase

The consistency checker works on different type of the domain knowledge: rule clusters, tables, and mathematical functions. For each of these knowledge types, we design different verification module.

1.2.1 Rule checker module

The main function of the rule checker is detecting consistency error of the rule cluster. Consistency in the rule cluster of the KB appears as: undefined concept, undefined property, undefined property values, duplicate rule pairs, conflict rule pairs, and subsumed rule pairs.

Syntax errors are frequent source of consistency errors. Detecting undefined concept, undefined property, undefined property values is realized by comparing each concept, property, and property value used in every rule against their corresponding definitions. Detecting duplicate, conflict, and subsumed rule pairs are realized by comparing each rule against every other rule within the same rule cluster.

1.2.1.1 Algorithm

Begin
    Get rule cluster R of the KB;
    I:=1; N:= number of rules of R;
    Con:= get all defined concept in the KB;
    While I <= N do
    Begin
        Get $rule_i$ *of R;*
        C_i *= get used concept of* $rule_i$*;*
        Diff:= Get difference between C_i *and Con;*
        If Diff $\neq \emptyset$ *then*
            Output error message

```
        J:=I+1;
        Begin
            While j<=n do
            Get rule_j of R;
            Compare rule_i with rule_j;
            J:=j+1;
        End
        I:=I+1;
    End
End
```

Hint to the implementer:

The comparison process is done be comparing each premise of the rule $rule_i$ with the premises of $rule_j$ in both the condition and action part. Three different flags are used to identify the result of the comparison. Same (S) is assigned to 1 when we found the same concept-property-value of $rule_i$ in $rule_j$. If we found the same concept-property of $rule_i$ but with different values in $rule_j$ then check whether this pair of type multiple then set D (different=1) or single set C(conflict=1). When this pair is not found set D(different=1). The result of the comparison process is calculated according to the following table:

Premise			Action			Result
S	C	D	S	C	D	
1	0	0	1	0	0	Duplicate
1	0	1	1	0	0	Subsumed
1	0	0	1	0	1	subsumed
1	1	0	1	0	0	subsumed
1	0	0	1	1	0	subsumed
1	0	0	1	1	0	conflict
1	0	0	0	1	0	conflict

1.2.2 Table checker module

The main function of the table checker is detecting consistency error of the table. Consistency in the table of the KB appears as: undefined concept, undefined property, undefined property values, duplicate table rows, and conflict table rows.

1.2.2.1 Algorithm

Begin
Get table T of the KB;
H:= get table header;
I:=1; N:= number of concept-property of H;
J:=1; Len:= Number of rows of T;
Con:= get all defined concept in the KB;
While I <= n do
Begin
C_i *=concept-property of H*
Get difference between C_i *and Con;*
i:=I+1;
End
While j <= Len do
Begin
row_j *=get row of T;*
check possible values of row_j*;*
k:=J+1;
Begin
While k<=n do
get row_k *of T*
Compare row_j *with* row_k*;*
k:=k+1;
End
j:=j+1;
End
End

1.2.3 Function checker module

The main function of the function checker is detecting consistency error of the function. Consistency in the function of the KB appears as: undefined concept, undefined property. Moreover, the function input/output concept-property should be only of type integer or real.

1.2.3.1 Algorithm

Begin
Get Function F of the KB;
Con:= get all defined concept in the KB;
CF = get used concept of F;
N:= Number of Input/output concept of F;
While I <= n do

```
    Begin
        Find out if CF[i] is defined in KB;
        Get type of CF[i];
        If type of CF[i] # real or integer then
            Output error message " incorrect concept-property type";
        I:=I+1;
    End
End
```

1.2.4 Creating the Domain Layer Table

Another function of the consistency checker phase is to create the domain table that is used to support the subsequent verification processes. This table contains the needed information about the use of every concept-property pairs in the domain layer. The basic idea behind constructing this table is to accelerate searching for any defined concept-property pair used in the domain layer which is heavily used in subsequent phases. This table consists of the following fields:

Relation name: The name of the domain layer component defined in the KB.

Relation type: The type of the domain layer component (rule cluster, table, and function).

Input property: The names of concept-property pairs used in the input part of the domain layer component.

Output property: The names of concept-property pairs used in the output part of domain layer component.

1.2.4.1 Algorithm

```
Begin
    Get all defined rule clusters Rs of the system;
    Get all defined Functions Fs of the system;
    Get all defined tables Ts of the system;
    For I=0 to I= number of Rs do
    Begin
        In=Get input concept-property pairs of Rs[i];
        Out=Get output concept-property pairs of Rs[i];
        Store name of Rs[i]in the domain knowledge name field of table;
        Store "rule" in the type field of the table;
        Store In  in the input field of table;
```

```
        Store Out  in the output field of table;
    End
    For I=0 to I= number of Ts do
    Begin
        In=Get input concept-property pairs of Ts[i];
        Out=Get output concept-property pairs of Ts[i];
        Store name of Ts[i] in the domain knowledge name field of table;
        Store "table" in the type field of the table;
        Store In  in the input field of table;
        Store Out  in the output field of table;
    End
    For I=0 to I= number of Fs do
    Begin
        In=Get input concept-property pairs of Fs[i];
        Out=Get output concept-property pairs of Fs[i];
        Store name of Fs[i] in the domain knowledge name field of table;
        Store function in the type field of the table;
        Store In  in the input field of table;
        Store Out  in the output field of table;
    End
End
```

1.3 Completeness Checker Phase

The purpose of the check for completeness phase is to scan the whole knowledge base looking for unfirable domain relation, and unused consequence. At CLAES we distinguishes between three types of value source: user when the property value is input by the user, database when the value is queried from a database, derived when the value is concluded by a domain knowledge component. The unfirable domain relation is detected when one of the used property in the input part of the domain relation has a defined source of value to be derived and the property does not appear in the output part of any other domain knowledge component. On the other hand, if the output part of any domain knowledge component is neither one of the final goals, nor it is used to fire other domain knowledge component then it is unused consequence.

1.3.1.1 Algorithm

```
Begin
    Tab:= Get domain layer table;
    Con:= get all defined concept in the KB;
    I:=1; N:= Number of rows of T;
```

```
        While I <= n  do
        Begin
           In := input column of Tab[i];
           Out:= output column of Tab[i];
           J:=1; Len:= length of In;
           k:=1; Len1:= length of Out;
           flag,flag1:=Boolean, initially=0;
           While j <= len  do
           begin
               if source of value of In[j] == drived then
               begin
                   Flag= find In[j] in the all output filed of tab;
                      If Flag == 0 then
                      begin
                             Get name and type of tab[i];
                             Output message " unfirable domain knowledge
                             component"
                      End
               end
               j:=j+1
           end

           While k <= len1  do
           begin
               if out[k] != goal then
               begin
                      Flag1 = find out[k] in the all input filed of tab;
                      If Flag1 == 0 then
                      begin
                             Get name and type of tab[i];
                             Output message " unusable consequence of that
                             domain knowledge component";
                      end
               end
           K:=k+1;
           end
        End
End
```

1.4 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 concept-property pairs to the output concept-property pairs of each defined domain knowledge component using the domain layer table.

Two main errors are detected during this phase: redundant paths, and circular paths. A redundant path is found when it is possible to reach the same conclusion from the same inputs through different paths. Circular paths are detected when a concept-property pair appears as an input of one domain knowledge component and as output concept-property pair of another domain knowledge component and a path between the other edges of these domain knowledge components can be reached.

1.4.1.1 Algorithm

Step1: create the domain knowledge graph

```
G:= empty graph
Tab:= domain knowledge table;
I:=1; N:= length of the domain knowledge table;
While I<=N do
Begin
     In := input column of Tab[i];
     Out:= output column of Tab[i];
     J:=1; Len:= length of In;
     k:=1; Len1:= length of Out;
     while j<= len do
        begin
           add node (g, In[j]);
           k:=1;
           while k<= len1 do
             begin
                    if !(node(g, out[k])) then
                           add node (g, out[k]);
                    add edge(g,In[j],out[k]);
                    k++;
             end
           j++;
        end
     I++;
end
```

Step2: detect circular paths

```
G:= domain knowledge graph;
L:= list containing all goal concept-property pairs;
N:= length of L; Len:= length of G;I:=1;j:=1;
While I<=N do
begin
    Y== L[i]
    Begin
        While j<= Len do
        Begin
            If Y == node (G, J)  then
            begin
                SG==Get all edges (G,J);
                K:=1; Len1:=length of SG;
                While adjac (SG,k) != Ø do
                Begin
                    Adj:= = adjac (SG,k);
                    If node(SG,Adj) == Y then
                        Output circular path for node(G,J);
                    K++;
                End
            End
        J++;
        End
    End
    I++;
end
```

Step3: detect redundant paths

```
G:= domain knowledge graph;
L:= list containing all goal concept-property pairs;
N:= length of L; Len:= length of G;I:=1;j:=1;
While I<=N do
begin
    Y== L[i]
    Begin
        While j<= Len do
        Begin
            If Y == node (G, J)  then
            begin
                SG==Get all edges (G,J);
                Compare edges of SG to get redundant paths
            End
```

```
        J++;
        End
    End
    I++;
end
```

2 Inference Layer Verification

The main functions of the inference layer verification are:

1.Create the inference table

2. Inference checker module.

2.1 Creating the inference table.

The step checker creates an inference table that is used to facilitate the detection of inference layer 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 (s) of the inference step.

static role: The list of the domain knowledge comonents that are used by this inference.

2.1.1 Algorithm

```
Begin
    ISTab:= Inference table;
    Get all defined inference step ISs of the system;
    For I=0 to I= number of ISs do
    Begin
        N=get name of the inference step;
        I= get defined input role of N;
        O= get defined output role of N;
        S= get defined static role of N;
        Store N in the inference name field of ISTab;
        Store I in the input role field of the ISTab;
        Store O  in the output role field of ISTab;
        Store S  in the static role field of ISTab;
    End
End
```

2.2 Step checker module

The main function of the step checker is detecting inference steps consistency error. Each inference step operates over data elements corresponding to the domain knowledge components. The input-role refers to a list of input data elements of the inference step. These elements correspond to a combination of the input-property of the domain knowledge components 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-property of the domain knowledge components which this inference uses. Inconsistency arises when the input/output-role refers to data element that is not defined in any domain layer component

Each inference step 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. 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 or be the final goal.

2.2.2 Algorithm

/*This algorithm is used to ensure that all used domain relation by each inference step are already defined in the domain layer*/

ISTab:= Inference table;

```
Tab:= domain knowledge table; N:= length of the inference table;
begin
        For I=1: I<=n do
        begin
                S:= static role field of ISTab[i]
                Len := number of relation in S;
                For j:=1 ;j<= Len do
                begin
                        If S[j] not appear in the relation name field of Tab then
                        Output message" undefined domain relation S[j] used in ISTab[j]"
                        J++;

                End

        I++
```

End

end

2.2.3 Algorithm

/* this algorithm is used to detect inference layer inconsistency*/

ISTab:= Inference table;

N:= length of the inference table;

```
begin
    For I=1: I<=n do
    Begin
        Out:= output role field of ISTab[i]
        Len := number of relation in Out;
        For j:=1; j<= Len do
        Begin
            If out[j] not appear in the inpu role field of ISTab or not final output then
            Output message" unused output role Out [j]"
            J++
        End
        I++
    End
end
```

3 KADS layers Verification

This verification activity aims at elimination of inconsistency errors arise due to the interactions between different knowledge layers. This happens when any of the three layers refers to undefined parts of another layer. 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 domain knowledge component of the domain layer. When one of the knowledge layers refers to undefined or erroneous parts of another layer, inconsistency between layers occurs.