Annals of the University of Craiova, Math. Comp. Sci. Ser.
Volume 35, 2008, Pages 149–170
ISSN: 1223-6934

An Extension of Inheritance Knowledge Bases and Computational Properties of their Answer Functions

Claudiu Popîrlan and Nicolae Ţăndăreanu

Abstract. In this paper a model of knowledge representation using the inheritance is proposed. In comparison with other models which use the inheritance mechanism, our model is endowed with the multiple inheritance and includes a parameter for each attribute value. This parameter specifies the extension treated in this paper. Several particular cases are exemplified. In particular the uncertain knowledge can be represented by this method. The concepts of interrogation, deduction and answer mapping are defined. The computational properties of the answer mapping are studied. We give a necessary and sufficient condition for the case when an infinite number of steps are performed by the answer mapping. The last section contains several open problems in connection with the subject treated in this paper.

2000 Mathematics Subject Classification. 68T30.
Key words and phrases. Inheritance, attribute, object, slot, strict partial order, interrogation, answer mapping, uncertainty, computability, decidability.

1. Introduction

Various aspects of the inheritance mechanism were studied. The language DLP is a knowledge representation language, which extends disjunctive logic programming (with strong negation) by inheritance ([4]). The inheritance of business rules in the medical insurance domain was studied in [9]. A natural model-theoretic semantics for inheritance in frame-based knowledge bases, which supports inference by inheritance as well as inference via rules was treated in [14]. The implications of the lattice theory to characterize the features of the answer mapping in knowledge systems based on inheritance were described in [10], [11] and [12]. An interrogation by voice of an inheritance based knowledge system is given in [13].

The objectives of this paper can be described as follows:

(1) A model of knowledge representation using the inheritance is proposed. In comparison with other models which use the inheritance mechanism, our model is endowed with the following features:
   - The multiple inheritance is allowed for the computation of an attribute value.
   - The value of an attribute can specify some parameter such that our representation includes the uncertain knowledge (fuzzy and probabilistic) or this parameter can identify a risk factor if this value is selected. In the last case it is not difficult to obtain a minimum risk or a maximum risk study for a given knowledge base.
   - In our model a given attribute can be inherited from more that one object. In this case a choice function is introduced to define some choice strategy and this function selects only one attribute from the set of all inherited attributes. Moreover, this function can specify a new attribute value which is obtained by means of the

Received: 19 December 2008.

selected values. In this manner the value of an attribute can be the inherited value, but can be a modified value also.
- An object can specify several values for a given attribute. In this case an object can inherit some of them or can obtain a new value by means of these values.

(2) The answer mapping is defined and its mathematical properties are studied. The main problem is connected by the computations of the values of this mapping in a finite number of steps. We prove that this is a decidable problem. This study is based on a necessary and sufficient condition for the case when an infinite number of steps are performed.

The results obtained in this paper allow to develop several research problems in the future:

(1) The decomposition of an inheritance knowledge base into disjoint components such each component is also an inheritance knowledge base. Each component can be uploaded on a work station in an network architecture. The interrogation can be performed "locally" on each station. The great knowledge bases can use such decomposition to improve the running time.
(2) We can use the mobile agents to perform the tasks of an inheritance based knowledge system: to interrogate, erase, insert and update a component and so on.

The paper is organized as follows. Section 2 contains a few results concerning the algebra of binary relations. In Section 3 the concepts of objects and inheritance are defined in an intuitive manner. Section 4 contains the concepts of inheritance knowledge base and accepted knowledge base. In Section 5 we explain what is an interrogation of an inheritance knowledge base. In Section 6 we define the mapping Val_{attr} which computes the value of an attribute for a given object. In Section 7 we study the finiteness of the computations for the mapping Val_{attr} and we give a necessary and sufficient condition for the infiniteness of the computations for this mapping. This study allows to treat a decidable problem in Section 8. In Section 9 we specify the classic case of the inheritance knowledge bases and we show that this is a simple particular case of the representation treated in the present paper. The last section, Section 10, contains the conclusions of our study and several ideas to develop this research line.

2. Prerequisites

In this section we recall the main mathematical concepts which are used in this paper.

A binary relation on the set X is a subset $\rho \subseteq X \times X$. The *transitive closure* of ρ is the least binary relation r on X such that $\rho \subseteq r$ and r is transitive. In other words, if $\overline{\rho}$ denotes the transitive closure of ρ then the following two conditions are satisfied:

(1) $\rho \subseteq \overline{\rho}$
(2) If $\rho \subseteq r \subseteq X \times X$ is a transitive relation then $\overline{\rho} \subseteq r$.

The *diagonal* of X is the binary relation

$$\Delta_X = \{(x, x) \mid x \in X\}$$

A binary relation ρ is a *strict partial order* on X if the following conditions are satisfied:

(1) $\Delta_X \cap \rho = \emptyset$
(2) $(x, y) \in \rho \Longrightarrow (y, x) \notin \rho$ for every $x, y \in X$
(3) $(x, y) \in \rho, (y, z) \in \rho \Longrightarrow (x, z) \in \rho$ for every $x, y, z \in X$

Comment 2.1. *The definition given above is the classical definition of a strict partial order ([8]). In fact, the second condition is a consequence of the other two conditions. Really, if by contrary we assume that $(x, y) \in \rho$ and $(y, x) \in \rho$ for some $x, y \in X$ then by transitivity we obtain $(x, x) \in \rho$ and this property shows that $\Delta_X \cap \rho \neq \emptyset$.*

The powers of the relation ρ are defined recursively as follows:

$$\begin{cases} \rho^1 = \rho \\ \rho^{n+1} = \rho^n \circ \rho, \ n \geq 0 \end{cases}$$

where $\circ$ is the usual product operation between binary relations,

$$\rho_1 \circ \rho_2 = \{(x, y) \in X \times X \mid \exists z \in X : (x, z) \in \rho_1, (z, y) \in \rho_2\}$$

The product operation is an associative one:

$$(\rho_1 \circ \rho_2) \circ \rho_3 = \rho_1 \circ (\rho_2 \circ \rho_3)$$

for every binary relations ρ_1, ρ_2 and ρ_3 over X. As a consequence, for the powers of a binary relation ρ we have

$$\rho^p \circ \rho^q = \rho^{p+q}$$

An element $x \in X$ is a *minimal element* with respect to ρ if

$$\{y \in X \mid (y, x) \in \rho, y \neq x\} = \emptyset$$

Notation 2.1. *The set of all minimal elements of the set X with respect to ρ is denoted by* $Min_\rho(X)$.

There is a graphical method to represent a binary relation ρ. In the most cases this representation gives an intuitive image of the properties (the symmetry, the reflexivity, the anti-symmetry, the transitivity etc) of the corresponding relation. To obtain this representation we draw a rectangle for each element of X and we write the name of the element in this rectangle. Then, we draw an arc from x to y if $(x, y) \in \rho$.

3. Objects and inheritance

The concepts of object and inheritance are the main concepts used in this paper. We briefly describe these concepts in this section.

In our vision an object is characterized by:

- The *name* of the object; it identifies uniquely the object.
- A finite set of symbolic names. Each of them designates another object that is named a *parent* of the object. A given object may contain zero or more parents.
- A finite set of *slots*; a slot is an ordered pair of the form (*attribute*, *value*), where *attribute* is the symbolic name of some feature and the *value* is its corresponding value.

The objects are virtually linked by the relation parent-child. If p is a parent of the object ob then ob is a *child* of p. A child has proper features and can *inherit* other features from its parents. Each feature of an object is given by an *attribute*. An attribute can specify some immediate value (for example the height, the color, the weight etc) or can identify the name of a method or procedure that computes the value of the attribute. Moreover, we suppose that some parameter is assigned to each value of an attribute. This parameter establishes the "greatness" of the corresponding value. We shall detail this aspect in this section.

Let us suppose we search the value V of some attribute A of the object F. Two cases are possible:

(1) The description of F contains at least one slot of the form (A, V, p) or (A, P, p), where V is an immediate value, P is the name of a procedure and p is a parameter associated to (A, V). We denote by $Slot(F)$ the set of all these slots. Intuitively we suppose that we have a choice strategy given by a mapping $Choice$ such that $Choice(Slot(F)) = (A, T, p)$ is the selected slot. If $T = V$ then the value of A is V and p is the parameter associated to (A, V). If $T = P$ then the value V is returned by the procedure P for some values of its arguments and p is the parameter associated to this value.

(2) The description of F does not contain any slot such that its first component is A. In this case the attribute value is obtained by means of some parent of F. In other words the corresponding attribute is *inherited* from its parents. This situation is iterated and this means that if each of these parents does not contain the corresponding attribute then we search it for the parents of the parents and so on.

We consider the following sets:

- L_{obj} is the set of the object names. Each element of L_{obj} can designate some object.
- L_{attr} is the language of all attribute names.
- V_{dir} is the set of all *direct* values of an attribute.
- L_{proc} is the language of all procedure names.
- $Param$ is a set of parameters.

Definition 3.1. *A* ***slot*** *is an element of the set $L_{attr} \times (V_{dir} \cup L_{proc}) \times Param$. An* ***object*** *is an element of the set*

$$L_{obj} \times 2^{L_{obj}} \times 2^{L_{attr} \times (V_{dir} \cup L_{proc}) \times Param}$$

It follows that an object is described by three components:

- The first component gives the *object name*.
- Every element of the second component is a "direct" parent of this object.
- The last component gives the slots of the object.

4. Inheritance knowledge bases

In this section we introduce the concepts of *inheritance knowledge base* and *accepted knowledge base*.

We consider a subset $K_0 \subseteq L_{obj} \times 2^{L_{obj}} \times 2^{L_{attr} \times (V_{dir} \cup L_{proc}) \times Param}$. If $x = (m, P, Q) \in K_0$ is an object then m determines uniquely the object x and we denote by $N(x) = m$ the name of x. For this reason an object is denoted by $x = (N(x), P_x, Q_x)$.

Definition 4.1. *An* ***inheritance knowledge base*** *is a pair $K = (Obj(K), \rho_K)$, where*

(1) *$Obj(K) \subseteq L_{obj} \times 2^{L_{obj}} \times 2^{L_{attr} \times (V_{dir} \cup L_{proc}) \times Param}$ is a finite set of elements named the* ***objects*** *of K, such that if $x = (N(x), P_1, Q_1) \in Obj(K)$, $y = (N(y), P_2, Q_2) \in Obj(K)$ and $N(x) = N(y)$ then $P_1 = P_2$ and $Q_1 = Q_2$.*

(2) *$\rho_K \subseteq Obj(K) \times Obj(K)$ is the* ***relation generated*** *by K, which is defined as follows:*

$$(x, y) \in \rho_K \Longleftrightarrow N(x) \in P_y$$

(3) *$\rho_K^i \cap \rho_K^j = \emptyset$ for $i \neq j$*

Remark 4.1. *We denote by $Proc(K)$ the set of all procedure names appearing in $Obj(K)$. We suppose that the set of procedure names and the set of object names of K are two disjoint sets. We denote by $Attr(K)$ the set of all attributes $a \in L_{attr}$ such that there is an object in $Obj(K)$ which contains the attribute a.*

Based on the first condition from Definition 4.1 an inheritance knowledge base K can not contain simultaneously the following information:

$(table, \{furniture, \}, \{(color, green, 10)\})$

$(table, \{furniture\}, \{(weight, 20, 40)\})$

In this case K contains the object

$(table, \{furniture\}, \{(color, green, 10), (weight, 20, 40)\})$

The third condition is imposed to avoid a situation presented in Figure 1, where ob_3 can be interpreted as the father and the grandfather of the same object ob_6. In this example we have $\rho_K^1 \cap \rho_K^2 \neq \emptyset$.

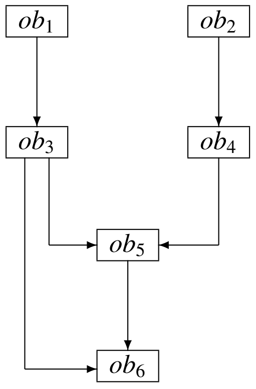

FIGURE 1. An example of inheritance

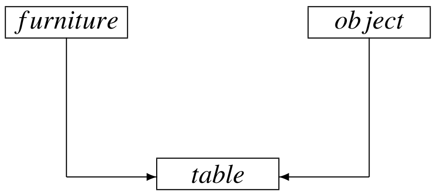

FIGURE 2. A multiple inheritance

Remark 4.2. *Let us suppose we have the following objects:*
$(table, \{furniture, object\}, \{(color, green, 10)\})$
$(furniture, \{\}, \{(weight, 20, 10)\})$
$(object, \{\}, \{(weight, 30, 50)\})$
The inheritance is shown in Figure 2. If this is the case then the attribute ***weight*** *for the object* ***table*** *can be inherited both from* ***furniture*** *and* ***object****. If the "maximum" strategy is used to select the attribute values then the value of the attribute weight for the object table is 30.*

Definition 4.2. *If* $(y, x) \in \rho_K^s$ *for some* $s \geq 1$ *then* y *is a* ***parent of order*** s *for* x.

Notation 4.1. *Let* $ob = (x, P_x, Q_x)$ *be an arbitrary object. For an arbitrary attribute name* $a \in L_{attr}$ *we write* $a \hookleftarrow\Box\, ob$ *(or* $a \hookleftarrow\Box\, x$*) if* Q_x *contains a slot* (a, v, p) *for some* $v \in V_{dir} \cup L_{proc}$.

Consider an arbitrary object ob. An intuitive aspect of the inheritance can be presented as follows. The attribute $a_1 \in L_{attr}$ **can be inherited** for ob from the object ob_1 if the following conditions are satisfied:
(1) $a_1 \not\hookleftarrow\Box\, ob$ and $a_1 \hookleftarrow\Box\, ob_1$;
(2) ob_1 is a parent of order $k + 1$ for ob ($k \geq 0$) and there is no parent ob_2 of order $j \leq k$ for ob such that $a_1 \hookleftarrow\Box\, ob_2$.
If there are r parents ($r \geq 2$) $b_1, \ldots, b_r$ of order $k + 1$ such that $a_1 \hookleftarrow\Box\, b_i$ for $i \in \{1, \ldots, r\}$ then the choice strategy is used to select one of them.

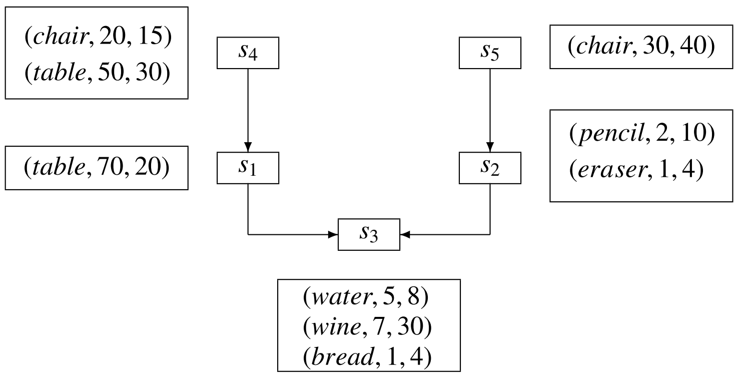

FIGURE 3. The knowledge base of Example 4.2

Example 4.1. *Let us consider the following objects:*

$(n_1, \{\}, \{(a_1, v_1, p_1)\})$
$(n_2, \{\}, \{(a_1, v_2, p_2)\})$
$(n_3, \{n_1\}, \{(b_1, q_1, p_3)\})$
$(n_4, \{n_1, n_2\}, \{(b_2, q_2, p_3)\})$
$(n_5, \{n_3\}, \{(c_1, q_3, p_4)\})$
$(n_6, \{n_4\}, \{(c_2, q_4, p_5)\})$

We observe that n_1 and n_2 are parents of order 2 for n_6. Moreover, we have the following properties:

$c_2 \hookleftarrow\!\Box\, n_6$, $b_1 \hookleftarrow\!\Box\, n_3$;
a_1 can be inherited for n_6 both from n_1 and n_2;
a_1 can be inherited for n_5 only from n_1.

Example 4.2. *Let us consider the shops $s_1, \ldots, s_5$. A client can buy from each shop the products specified in Figure 3: **chair, table, pencil, eraser, water, wine** and **bread**. If an attribute is not specified for some shop s_i, this attribute can be inherited from the nearest parents of s_i. The objects s_4 and s_5 are parents of order 2 for s_3. The second component of an attribute is the cost of the product specified as the first component. Suppose we use the "maximum" strategy for parameters (this is a natural choice if the parameter specifies the quality of the product). In this case the value of the attribute **chair** for s_3 is 30.*

Definition 4.3.
*The relation $inh_K = \bigcup_{n\geq 1} \rho_K^n$ is named the **inheritance relation generated** by K.*

Proposition 4.1. *The relation inh_K is the transitive closure of the relation ρ_K.*

Proof. Let us verify that inh_K is transitive. Suppose that $(x, y) \in inh_K$ and $(y, z) \in inh_K$. There are $p \geq 1$ and $q \geq 1$ such that $(x, y) \in \rho_K^p$ and $(y, z) \in \rho_K^q$. But $(x, z) \in \rho_K^p \circ \rho_K^q = \rho_K^{p+q} \subseteq inh_K$. It follows that inh_K is a transitive relation. Let us prove that inh_K is the least transitive relation that contains ρ_K. From its definition we have $\rho_K \subseteq inh_K$. Suppose that

- $\theta \subseteq Obj(K) \times Obj(K)$
- $\rho_K \subseteq \theta$
- θ is a transitive relation

Let us verify that $\rho_K^p \subseteq \theta$ for every $p \geq 1$. For $p = 1$ this sentence is true. Suppose that the sentence is true for p. We have

$$\rho_K^{p+1} = \rho_K^p \circ \rho_K \subseteq \theta \circ \theta$$

But $\theta \circ \theta \subseteq \theta$ because θ is transitive. As a consequence we obtain $\rho_K^{p+1} \subseteq \theta$ and therefore $inh_K = \bigcup_{q \geq 1} \rho_K^q \subseteq \theta$. ■

Definition 4.4. *Let K be an inheritance knowledge base. An element $x \in Obj(K)$ is a* ***useless object*** *if $\{y \mid (x,y) \in \rho_K\} \cup \{y \mid (y,x) \in \rho_K\} = \emptyset$. We denote by* ***Useless(K)*** *the set of all useless objects from K.*

Proposition 4.2. *If $x \in Obj(K)$ is a useless object then $\{y \mid (x,y) \in inh_K\} \cup \{y \mid (y,x) \in inh_K\} = \emptyset$.*

Proof. Suppose by contrary that $\{y \mid (x,y) \in inh_K\} \cup \{y \mid (y,x) \in inh_K\} \neq \emptyset$. In order to make a choice we suppose that there is $y \in Obj(K)$ such that $(x,y) \in inh_K$. Taking into consideration the definition of inh_K it follows that there is a natural number $m \geq 1$ such that $(x,y) \in \rho_K^m$. Using the definition of ρ_K^m we deduce that there are $y_1, \ldots, y_m \in Obj(K)$ such that $(x,y_1) \in \rho_K$, $(y_i, y_{i+1}) \in \rho_K$ for $i = 1, \ldots, m$ and $y_{m+1} = y$. It follows that $(x,y_1) \in \rho_K$, which is not possible by our assumption. ■

The previous proposition shows that an useless object can not inherit attributes and can not send attributes to other objects.

Definition 4.5. *An inheritance knowledge base K is an* ***accepted base*** *if the following conditions are fulfilled:*

$$Useless(K) = \emptyset \tag{1}$$

$$Min_{inh_K}(Obj(K)) \neq \emptyset \tag{2}$$

$$inh_K \text{ is a strict partial order} \tag{3}$$

Proposition 4.3. *$Min_{inh_K}(Obj(K)) = Min_{\rho_K}(Obj(K))$*

Proof. Take an element $x \in Min_{inh_K}(Obj(K))$. There is no element $y \in Obj(K)$ such that$(y,x) \in inh_K$. Suppose that $x \notin Min_{\rho_K}(Obj(K))$. There is $z \in Obj(K)$ such that $(z,x) \in \rho_K$. But $\rho_K \subseteq inh_K$ and therefore $(z,x) \in inh_K$. By our assumption this is not possible because in this case x is not a minimal element with respect to inh_K. Conversely, suppose that $x \in Min_{\rho_K}(Obj(K))$. Suppose that $x \notin Min_{inh_K}(Obj(K))$. There is $z \in Obj(K)$ such that $(z,x) \in inh_K$. It follows that there are $y_1, \ldots, y_m \in Obj(K)$ such that $y_1 = z$, $y_m = x$ and $(y_i, y_{i+1}) \in \rho_K$ for $i \in \{1, \ldots, m-1\}$. It follows that x is not a minimal element with respect to ρ_K because $(y_{m-1}, x) \in \rho_K$. ■

As a consequence the condition (2) can be replaced by the following condition

$$Min_{\rho_K}(Obj(K)) \neq \emptyset$$

The relation inh_K gives a mathematical representation of the main relation between the objects of an inheritance knowledge base. We relieve the following aspects:

(1) An object (x, P_x, Q_x) of K is a *free of parents object* if $P_x = \emptyset$. Obviously an object of K is a minimal element with respect to ρ_K if and only if it is a free of parents object. The equation (2) requires for an accepted base to have at least one free of parents object.

(2) In particular, the condition (3) specifies the natural conditions given in the following sentences:
- an object can not be its parent: $x \notin P_x$ for every $x \in Obj(K)$

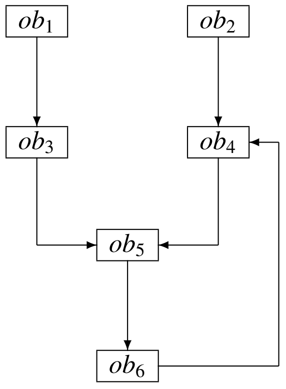

FIGURE 4. An example of inheritance

- if $x \in P_y$ then $y \notin P_x$ and more generally, if x is a parent of some order for y then y can not be a parent for x. To exemplify such a situation we consider the inheritance represented in Figure 4. In this case ob_4 is a parent of order 1 for ob_5 and ob_5 is a parent of order 2 for ob_4.

Remark 4.3. *Frequently we are interested to compute all the parents (of any order) for a given object of an inheritance knowledge base. The relation inh_K generated by an inheritance knowledge base allows to compute these parents because the parents of $x \in Obj(K)$ is the set $\{y \in Obj(K) \mid (y, x) \in inh_K\}$.*

In what concerns the computation of the relation inh_K we have the property stated in the next proposition.

Proposition 4.4. *Denote $\theta_n = \bigcup_{p=1}^{n} \rho_K^p$ for every $n \geq 1$. There is $s \geq 1$ such that $\theta_1 \subset \ldots \subset \theta_s = \theta_{s+1}$ and $inh_K = \theta_s$.*

Proof. We have $\theta_{p+1} = \theta_p \cup \rho_K^{p+1}$ and $\theta_p \subseteq 2^{Obj(K) \times Obj(K)}$. The last set is a finite one therefore there is $s \geq 1$ such that $\theta_1 \subset \ldots \subset \theta_s = \theta_{s+1}$. From $\theta_{s+1} = \theta_s$ and $\theta_{s+1} = \theta_s \cup \rho_K^{s+1}$ we deduce that $\theta_s \cup \rho_K^{s+1} = \theta_s$. This property shows that $\rho_K^{s+1} \subseteq \theta_s$.
Let us verify by induction on t that $\rho_K^{s+t} \subseteq \theta_s$ for every $t \geq 1$. For $t = 1$ the property is true as we have seen. Suppose that $\rho_K^{s+t} \subseteq \theta_s$. We have $\rho_K^{s+t+1} = \rho_K^{s+t} \circ \rho_K \subseteq \theta_s \circ \rho_K$. But $\theta_s \circ \rho_K = (\rho_K^1 \cup \ldots \cup \rho_K^s) \circ \rho_K = \rho_K^2 \cup \ldots \cup \rho_K^{s+1} \subseteq \rho_K^1 \cup \rho_K^2 \cup \ldots \cup \rho_K^{s+1} = \theta_{s+1} = \theta_s$.
Now, based on the definition of inh_K we obtain

$$inh_K = \theta_s \cup \bigcup_{i \geq 1} \rho_K^{s+i} = \theta_s$$

because $\rho_K^{s+i} \subseteq \theta_s$ for every $i \geq 1$. Thus the proposition is proved. ■

Remark 4.4. *In the remainder of this paper by a "knowledge base" we understand an accepted inheritance knowledge base.*

5. Interrogation of an inheritance knowledge base

The interrogation process and the corresponding answers are main operations for every knowledge base system. This is due to the fact that every communication interface between user and system takes into consideration both the interrogation process and the answer to an interrogation. The initial step is given by interrogation. The final step is given by the answer to an interrogation. Between these steps there are a lot of other steps which take into consideration the deduction process.

In this section we define the concepts of interrogation and answer to an interrogation and we specify some aspects connected by these concepts.

As we have seen the knowledge representation based on inheritance encapsulates the features of an object into some entity that becomes an element of a knowledge base. These features are represented by attributes. An attribute can designate a specific value of an attribute or can specify the name of a procedure that can compute the value of the corresponding attribute. An entity of a knowledge base includes also the information concerning the inheritance. It follows that there are two main problems connected by the interrogation of a knowledge base:

- Obtain the objects that can cooperate to obtain some property of a given object.
- Find the value of an attribute for a given object.

We consider an accepted knowledge base K and we recall that the inheritance relation generated by K is $inh_K = \bigcup_{p\geq 1}\rho_K^p$. For every $(x,a) \in Obj(K) \times Attr(K)$ and $p \geq 1$ we consider the set

$$Parent_K^p(x,a) = \{\, y \in Obj(K) \mid (y,x) \in \rho_K^p, a \hookleftarrow\!\Box\; y \,\} \tag{4}$$

of all parents of order p for x that contain the attribute a.

Definition 5.1. *The **order** of the attribute a_1 with respect to $x \in Obj(K)$ is defined as follows:*

$$ord_x(a_1) = \begin{cases} min\{p \mid Parent_K^p(x,a_1) \neq \emptyset\} \quad \texttt{if} \quad \bigcup_{p\geq 1} Parent_K^p(x,a_1) \neq \emptyset \\ \\ 0 \quad \texttt{if} \quad a_1 \hookleftarrow\!\Box\; x \\ \\ undefined \quad \texttt{otherwise} \end{cases}$$

Intuitively, if $ord_x(a_1)$ is defined and $ord_x(a_1) = s$ for some $s \geq 1$ then the set $Parent_K^s(x,a_1)$ gives the nearest parents of x which contain the attribute a_1. If $ord_x(a_1) = 0$ then the attribute a_1 can be taken just from the object x.

As we have seen the value of an attribute can be the value given by a procedure. We stipulate here the following assumptions concerning an arbitrary procedure name $p \in Proc(K)$:

- The procedure p has the formal arguments specified in a vector $Arg(p) = (b_1,\ldots,b_r)$, where $b_1,\ldots,b_r \in Attr(K)$.
- In order to call the procedure p we use some vector $(v_1,\ldots,v_r)$ of actual arguments, where each v_i is the value of the attribute b_i. This means that $v_i \in V_{dir} \times Param$.
- We denote by $p(v_1,\ldots,v_r)$ the value returned by p for the actual arguments $v_1,\ldots,v_r$.
- We shall suppose that p is given by a correct algorithm. Particularly This means that no running error can appear if $v_1,\ldots,v_r$ belong to the domain of p (for example division by zero, overflow or underflow operation etc).
- As we specified above the value of an actual argument is the value of an attribute. Two cases are possible:
  - At the time of procedure call the value of the attribute b_i is unknown. This case is encountered when there is no sufficient information to compute the value of the corresponding attribute, but if we update the knowledge base then this value can be computed. In this case we consider the value of the actual parameter $v_i = unknown$, without any parameter.
  - The value of the actual argument can not be computed. This case is treated in a separate section of this paper, where we develop the computability aspects. We

suppose that in this case we have $v_i = error$, if b_i is the attribute whose value can not be computed. No parameter is associated to the value *error*.

- We suppose that:
  - If $v_i \in V_{dir} \times Param$ for $i = 1, \ldots, r$ then $p(v_1, \ldots, v_r) \in V_{dir} \times Param$.
  - If there is $i \in \{1, \ldots, r\}$ such that $v_i = unknown$ and $v_j \neq error$ for every $j \neq i$ then $p(v_1, \ldots, v_r) = unknown$.
  - If there is $i \in \{1, \ldots, r\}$ such that $v_i = error$ then $p(v_1, \ldots, v_r) = error$.

Definition 5.2. *An* ***interrogation*** *of a knowledge base K is an element of the set* $Obj(K) \times Attr(K)$*. The* ***answer*** *of an interrogation* (x, a_1) *is the value of the attribute* a_1 *for* $x \in Obj(K)$*.*

As we shall see the answer can not be computed for any interrogation. Intuitively, the value *error* of an interrogation shows that the value of the attribute can not be computed. In the next paragraph we details this aspect.

6. The value of an attribute

In this section we define the mapping

$$Val_{attr} : Obj(K) \times Attr(K) \longrightarrow (V_{dir} \times Param) \cup \{unknown, error\}$$

which computes the value $Val_{attr}(x, a_1)$ of the attribute a_1 for x.

For a given pair $(x, a_1) \in Obj(K) \times Attr(K)$ we define:

$$Attr^0_K(x, a_1) = \{(v, p) \mid (a_1, v, p) \in Q_x\}$$

and for $s \geq 1$

$$Attr^s_K(x, a_1) = \{(v, p) \mid y \in Parent^s_K(x, a_1), (a_1, v, p) \in Q_y\}$$

Intuitively, $Attr^s_K(x, a_1)$ specifies all values of the attribute a_1 for x and their parameters, which are contained by the parents of order s.

We consider a mapping

$$Choice : 2^{(V_{dir} \cup Proc(K)) \times Param} \longrightarrow (V_{dir} \cup Proc(K)) \times Param$$

This mapping is used to choose or to obtain some attribute value from a set which contains several attribute values.

Example 6.1. *For example, if we denote* $U = \{(v_i, p_i)\}_{i=1}^n \subseteq (V_{dir} \cup Proc(K)) \times Param$ *then we can take*

$$Choice(U) = \begin{cases} (v_j, p_j) & \texttt{if} \quad 2 * p_j > \sum_{i=1}^n p_i \\ \\ (\sum_{i=1}^n v_i/n, \sum_{i=1}^n p_i/n) & \texttt{otherwise} \end{cases}$$

Remark 6.1. *We observe that the mapping Choice can be used not only to select some attribute value from a given set of attribute values, but also it can specify a new attribute value and/or a new parameter obtained by means of the corresponding values.*

We define the mapping $h : Obj(K) \times Attr(K) \longrightarrow ((V_{dir} \cup Proc(K)) \times Param) \cup \{no\}$ by

$$h(x, a_1) = \begin{cases} Choice(Attr^s_K(x, a_1)) & \texttt{if} \quad ord(x, a_1) = s \geq 0 \\ \\ no \quad \texttt{otherwise} \end{cases}$$

Intuitively $h(x, a_1) \neq no$ specifies the following entities:

- the value of a_1 for x or a method by means of which the value of a_1 for x can be obtained
- the parameter of this value

The value $h(x, a_1) = no$ shows that no parent of x contains a slot of the form (a_1, v_1, p_1) with respect to the attribute a_1.

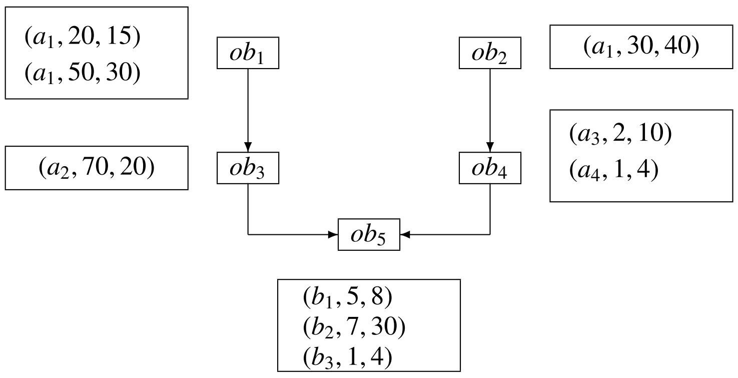

FIGURE 5. The case given in Example 6.2

Example 6.2. *Let us consider the case presented in Figure 5. We remark that we have two slots for the same attribute a_1 given in the definition of the object ob_1. We obtain:*
- $ord_{ob_5}(a_1) = 2$
- $h(ob_5, a_1) = Choice(\{(20, 15), (50, 30), (30, 40)\})$

If we take the function Choice from Example 6.1 then we obtain
$h(ob_5, a_1) = (100/3, 85/3)$
Let us consider that the knowledge base specified in Figure 5 contains also the object $ob_6 = (N(ob_6), \{N(ob_4)\}, \{(a_5, 10, 20))\}$. In this case we obtain $h(ob_5, a_5) = no$.

We consider a mapping $adj : Param \times Param \longrightarrow Param$. This mapping establishes a "matching rule" for parameters. For example,

$$adj(q_1, q_2) = (q_1 + q_2)/2$$

or

$$adj(q_1, q_2) = max\{q_1, q_2\}$$

combines 2 parameters to obtain a new value of parameter. The first expression gives the "average rule" and the second gives the "maximum rule".

Definition 6.1. *We define the mapping*

$$Val_{attr} : Obj(K) \times Attr(K) \longrightarrow (V_{dir} \times Param) \cup \{unknown, error\}$$

as follows:
- *If $h(x, a_1) = (v_1, q_1)$ and $v_1 \in V_{dir}$ then $Val_{attr}(x, a_1) = (v_1, q_1)$.*
- *If $h(x, a_1) = (p_1, q_1)$ and $p_1 \in Proc(K)$ then we consider the following two cases:*
  1. *If $Arg(p_1) = (b_1, \ldots, b_r)$ and $Val_{attr}(x, b_1) = (v_1, q_1)$, …, $Val_{attr}(x, b_r) = (v_r, q_r)$ are elements of $V_{dir} \times Param$ then*
  $$Val_{attr}(x, a_1) = (u, q) \tag{5}$$
  where $p_1(Val_{attr}(x, b_1), \ldots, Val_{attr}(x, b_r)) = (u, s)$ and $q = adj(q_1, s)$.
  2. *Otherwise $Val_{attr}(x, a_1) = error$.*
- *If $h(x, a_1) = no$ then $Val_{attr}(x, a_1) = unknown$.*

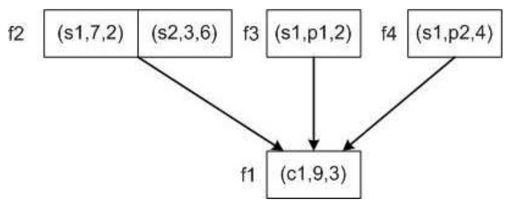

FIGURE 6. A part of a knowledge base

Remark 6.2. *We observe that the mapping Val_{attr} is a recursive one. The equation (5) shows that $Val_{attr}(x, a_1)$ is computed by means of the values $Val_{attr}(x, b_1), \dots, Val_{attr}(x, b_r)$.*

Definition 6.2. *Let K be an accepted knowledge base and $(x, a) \in Obj(K) \times Attr(K)$. We write*

$$K \vdash (x, a)$$

if and only if $Val_{attr}(x, a) \in (V_{dir} \times Param) \cup \{unknown\}$

Let us consider a part of a knowledge base K represented in Figure 6. We suppose that $p1, p2 \in Proc(K)$. We remark that $Attr_K^1(f_1, s_1) = \{(7, 2), (p_1, 2), (p_2, 4)\}$ and therefore we have here an example by which the value of an attribute can be a direct value or can be computed by two procedures. We have $ord_{f_1}(s_1) = 1$ and $h(f_1, a_1) = Choice(Attr_K^1(f_1, s_1))$. If we use the maximum strategy for parameters then $Val_{attr}(f1, s1)$ is computed by the procedure p_2.

Remark 6.3. *Various features can be specified by means of an attribute. In the most cases an attribute of an object specifies some feature of the corresponding object. Sometimes several distinguished aspects can be relieved by the inheritance representation. Let us take the following example: Peter and Helen look at television set, where they see a soccer player. Peter believes that the height of the player is 172cm, but Helen thinks that the height is 180cm. We can use the certainty factors to represent this knowledge piece. Suppose that we take the certainty factors from the real interval* $[0, 100]$. *Taking into consideration the uncertainty, Peter believes that the height of the player is 172cm with certainty factor 70. Helen believes that the height is 180 with certainty factor 50. These aspects can be represented by considering the following objects:*

$(Peter, \{\}, \{(height, 172, 70)\})$
$(Helen, \{\}, \{(height, 180, 50)\})$
$(player, \{Peter, Helen\}, \{\})$

Let us suppose that $Choice(\{(v_1, q_1), \dots, (v_r, q_r)\}) = (v_1 + \dots + v_r)/r, (q_1 + \dots + q_r)/r$. We obtain $Val_{attr}(player, height) = (176, 60)$. We remark that the attribute height of the object Peter specifies the height observed by Peter. Similar we have the attribute height of the object Helen. An inheritance based knowledge system contains several components. One of them refers to the communication between user and system. This component sends to the user the correct message after the computation of the value Val_{attr}. For example, $Val_{attr}(Peter, height) = (172, 70)$. As a consequence the system sends the message "Peter observed that the player's height is 172 with certainty factor 70" instead of "The height of Peter is 172 with certainty factor 70".

As we can observe from Definition 6.1 there are two central problems:
- the computation of the entity $h(x, a_1)$;
- the characterization of the case "error" for the mapping Val_{attr}; this problem is treated in the next section.

In order to compute the value $h(x,a_1)$ we use the set $Parent^s_K(x,a_1)$. The next proposition can be used to compute this set by a simple method.

Proposition 6.1. *Let us consider $(x,a_1) \in Obj(K) \times Attr(K)$. The sequences $\{S_i\}_{i\geq 0}$ and $\{Q_i\}_{i\geq 1}$ defined by $S_0 = \{x\}$,*

$$\begin{cases} S_{i+1} = \{y \mid (y,q) \in \rho^1_K, q \in S_i\} \\ \\ Q_{i+1} = \{q \in S_{i+1} \mid a_1 \hookleftarrow\!\Box\; q\} \end{cases}$$

for every $i \geq 0$, satisfy the following properties:
(1) *There is $n_0 \geq 0$ such that $S_i \neq \emptyset$ for every $i \leq n_0$ and $S_i = \emptyset$ for every $i \geq n_0 + 1$.*
(2) *For every $i \geq 0$ we have $S_{i+1} = \{y \mid (y,x) \in \rho^{i+1}_K\}$ and $Q_{i+1} = Parent^{i+1}_K(x,a_1)$.*

Proof. We verify by induction on $i \geq 0$ that

$$S_{i+1} = \{y \mid (y,x) \in \rho^{i+1}_K\} \tag{6}$$

For $i = 0$ the property is obtained directly from the definition of S_1. Suppose the property is true for i. From the definition of S_{i+2} we have

$$S_{i+2} = \{y \mid (y,q) \in \rho^1_K, q \in S_{i+1}\} \tag{7}$$

Take $y \in S_{i+2}$. There is $q \in S_{i+1}$ such that $(y,q) \in \rho^1_K$. By the inductive assumption we have $(q,x) \in \rho^{i+1}_K$, therefore $(y,x) \in \rho^1_K \circ \rho^{i+1}_K = \rho^{i+2}_K$. Thus we have

$$S_{i+2} \subseteq \{y \mid (y,x) \in \rho^{i+2}_K\}$$

In order to prove the converse inclusion we take an arbitrary element y such that $(y,x) \in \rho^{i+2}_K$. But $\rho^{i+2}_K = \rho^1_K \circ \rho^{i+1}_K$. It follows that there is q such that $(y,q) \in \rho^1_K$ and $(q,x) \in \rho^{i+1}_K$. By the inductive assumption we have $q \in S_{i+1}$. From (7) we obtain $y \in S_{i+2}$ and thus

$$S_{i+2} \supseteq \{y \mid (y,x) \in \rho^{i+2}_K\}$$

Now the relation $Q_{i+1} = Parent^{i+1}_K(x,a_1)$ is obviously true if we take into consideration (4), (6) and the definition of S_{i+1}.
We observe that $S_i = \emptyset$ for some $i \geq 1$. By contrary, suppose that $S_i \neq \emptyset$ for every $i \geq 1$. Because $S_i \subseteq Obj(K)$ and $Obj(K)$ is a finite set we deduce that there are $i \geq 1$ and $m \geq 1$ such that $S_i \cap S_{i+m} \neq \emptyset$. Using (6) we deduce that $\rho^i_K \cap \rho^{i+m}_K \neq \emptyset$, which is not possible in virtue of Definition 4.1. Thus we can take n_1 the least natural number i for which $S_i = \emptyset$ and then $n_0 = n_1 - 1$. Now it is obvious from definition that $S_i = S_{i+1}$ for every $i \geq n_0 + 1$ and the proposition is proved. ■

Remark 6.4. *Based on Proposition 6.1 it is not difficult to compute the set $Parent^n_K(x,a_1)$ for $n \geq 1$ because $Parent^n_K(x,a_1) = Q_n$. Really, the last element of the sequence S_0, S_1, Q_1, ..., S_n, Q_n is the set $Parent^n_K(x,a_1)$.*

7. The error value of Val_{attr}

In this section we study the case $Val_{attr}(x,a_1) = error$ in Definition 6.1. More precisely we give a necessary and sufficient condition to obtain this value.

Notation 7.1. *If $(x,a) \in Obj(K) \times Attr(K)$ then we write $(x,a) \in C_K(proc)$ to specify that the value of the attribute a for x is computed by some procedure.*

Consider an element $(x,a) \in Obj(K) \times Attr(K)$ such that $h(x,a)$ is defined, $h(x,a) = (p,q)$, where $p \in Proc(K)$. Shortly we denote

$$(x,a) \in C_K(proc)$$

to specify that the value of the attribute a for x is computed by a procedure. The procedure p uses some formal arguments, which are attribute names. As we specified in a previous section we considered the vector $(b_1, \ldots, b_r)$ of these attributes and we denoted $Arg(p) = (b_1, \ldots, b_r)$. In order to relieve the use of these attributes for the computation of $Val_{attr}(x, a)$ we denote also $arg(x, a) = Arg(p)$. In other words, if the value of the attribute $a \in Attr(K)$ for $x \in Obj(K)$ is computed by a procedure then $arg(x, a)$ gives the vector of the attributes whose values for x are passed to this procedure as actual arguments. Simply we write $b \sqsubseteq arg(x, a)$ if b is a component of the vector $arg(x, a)$.

Remark 7.1. *From Definition 6.1 we have the following two cases:*
(1) *If $(x, a) \notin C_K(proc)$ then $Val_{attr}(x, a) \in (V_{dir} \times Param) \cup \{unknown\}$.*
(2) *If $(x, a) \in C_K(proc)$ then $Val_{attr}(x, a) \in (V_{dir} \times Param) \cup \{error\}$.*

Definition 7.1. *We define the mapping $F : C_K(proc) \longrightarrow 2^{Attr(K)}$ as follows*

$$F(x, a) = \{b \in Attr(K) \mid (x, b) \in C_K(proc), b \sqsubseteq arg(x, a)\}$$

This definition shows that $F(x, a)$ gives the **set** of attribute names whose values are computed by procedures and their values are used to compute the value of a for x. For example, if $arg(x, a) = (b, c, b)$ then $F(x, a) = \{b, c\}$.

Definition 7.2. *If $(x, a) \in C_K(proc)$ then we write $a \Rightarrow_x b$ if and only if $b \in F(x, a)$. The transitive closure of this relation is denoted by $\Rightarrow_x^+$. We denote $[a]_x = \{b \mid a \Rightarrow_x^+ b\}$. The binary relation $\Rightarrow_x^+$ is named* ***derivation****. We denote $a \Rightarrow_x^n b$ if there are b_0, b_1, ..., b_n such that $b_0 = a$, $b_n = b$ and $b_i \Rightarrow b_{i+1}$ for every $i \in \{0, \ldots, n-1\}$.*

Proposition 7.1. *The following properties are satisfied:*
(1) *If $c \in [b]_x$ and $b \in [a]_x$ then $c \in [a]_x$.*
(2) *$[a]_x = \bigcup_{n \geq 1} Z_n^{(x,a)}$, where the sequence $\{Z_n^{(x,a)}\}_{n \geq 0}$ is recursively defined as follows:*

$$\begin{cases} Z_0^{(x,a)} = \{a\} \\ \\ Z_{n+1}^{(x,a)} = \{b \mid \exists t \in Z_n^{(x,a)}, t \Rightarrow_x b\},\ n \geq 0 \end{cases} \tag{8}$$

(3) *$[a]_x = \bigcup_{n \geq 1} W_n^{(x,a)}$, where the sequence $\{W_n^{(x,a)}\}_{n \geq 0}$ is defined as follows:*

$$\begin{cases} W_0^{(x,a)} = \{a\} \\ \\ W_{n+1}^{(x,a)} = \{b \mid a \Rightarrow_x^{n+1} b\},\ n \geq 0 \end{cases} \tag{9}$$

Proof. If $c \in [b]_x$ then there is a derivation $b \Rightarrow_x^+ c$. Similarly, if $b \in [a]_x$ then there is a derivation $a \Rightarrow_x^+ b$. But the relation $\Rightarrow_x^+$ is transitive, therefore $a \Rightarrow_x^+ c$. Thus the first part of the proposition is proved.
In order to prove the second part we prove first that for every $n \geq 1$

$$Z_n^{(x,a)} = \{b \mid a \Rightarrow_x^n b\} \tag{10}$$

We verify (10) by induction on n. For $n = 1$ from (8) we obtain:

$$Z_1^{(x,a)} = \{b \mid a \Rightarrow_x b\}$$

and thus (10) is true for $n = 1$. Suppose that (10) is true for some natural number n. From (8) we obtain

$$Z_{n+1}^{(x,a)} = \{b \mid \exists t \in Z_n^x, t \Rightarrow_x b\}$$

The following sentences are equivalent:
(1) $b \in Z_{n+1}^{(x,a)}$

(2) there is $t \in Z_n^x$ such that $t \Rightarrow_x b$
(3) there is a derivation $a \Rightarrow_x t$ such that $t \Rightarrow_x b$
(4) there is a derivation $a \Rightarrow_x^{n+1} b$
therefore (10) is true for $n + 1$.
From (10) and (9) we have $Z_n^{(x,a)} = W_n^{(x,a)}$ for every $n \geq 0$.
Let us verify that $[a]_x = \bigcup_{n\geq 1} Z_n^{(x,a)}$. Take $b \in [a]_x$. From Definition 7.2 we deduce that there is a natural number n such that $a \Rightarrow_x^n b$. By (10) we have $b \in Z_n^{(x,a)}$. Thus $[a]_x \subseteq \bigcup_{n\geq 1} Z_n^{(x,a)}$. The converse implication is immediate. It follows that $[a]_x = \bigcup_{n\geq 1} Z_n^{(x,a)}$. Now the relation $[a]_x = \bigcup_{n\geq 1} W_n^{(x,a)}$ is also proved because $Z_n^{(x,a)} = W_n^{(x,a)}$ for every $n \geq 1$. ■

Proposition 7.2. *The sequence* $\{Z_n^{(x,a)}\}_{n\geq 0}$ *satisfies one and only one of the following two conditions:*

(i) There is $k \geq 0$ *such that* $Z_i^{(x,a)} \neq \emptyset$ *for* $i \in \{0, \dots, k\}$ *and* $Z_i^{(x,a)} = \emptyset$ *for* $i \geq k + 1$.

(ii) $Z_n^{(x,a)} \neq \emptyset$ *for every* $n \geq 0$.

Proof. Obviously either $Z_n^{(x,a)} \neq \emptyset$ for every $n \geq 0$ or there is $p \geq 1$ such that $Z_p^{(x,a)} = \emptyset$. If the second case is encountered then denote by s the least p satisfying this property. From (8) we deduce that $Z_{s+1}^{(x,a)} = \emptyset$ and by induction on m we can verify that $Z_{s+m}^{(x,a)} = Z_s^{(x,a)} = \emptyset$. ■

Proposition 7.3. *If there is* $b \in \bigcup_{n\geq 0} Z_n^{(x,a)}$ *such that* $b \in [b]_x$ *then* $Z_n^{(x,a)} \neq \emptyset$ *for every* $n \geq 0$.

Proof. Suppose that there is $k \geq 0$ such that $b \in Z_k^{(x,a)}$ and $b \in [b]_x$. By (8) we have $[b]_x = \bigcup_{n\geq 1} Y_n^{(x,b)}$, where

$$\begin{cases} Y_0^{(x,b)} = \{b\} \\ \\ Y_{n+1}^{(x,b)} = \bigcup_{t \in Y_n^{(x,b)}} F(x,t), \ n \geq 0 \end{cases}$$

We prove by induction on $s \geq 0$ that

$$Y_s^{(x,b)} \subseteq Z_{k+s}^{(x,a)} \tag{11}$$

Really, we have:

- $Y_0^{(x,b)} \subseteq Z_k^{(x,a)}$ because $Y_0^{(x,b)} = \{b\}$ and $b \in Z_k^{(x,a)}$.
- Suppose that $Y_s^{(x,b)} \subseteq Z_{k+s}^{(x,a)}$. But

$$Y_{s+1}^{(x,b)} = \bigcup_{t \in Y_s^{(x,b)}} F(x,t) \tag{12}$$

and

$$Z_{k+s+1}^{(x,a)} = \bigcup_{t \in Z_{k+s}^{(x,a)}} F(x,t) \tag{13}$$

By the inductive assumption we have $Y_s^{(x,b)} \subseteq Z_{k+s}^{(x,a)}$ therefore from (12) and (13) we deduce $Y_{s+1}^{(x,b)} \subseteq Z_{k+s+1}^{(x,a)}$.

From $b \in [b]_x$ and $[b]_x = \bigcup_{n\geq 1} Y_n^{(x,b)}$ we deduce that there is $p \geq 1$ such that $b \in Y_p^{(x,b)}$. Using (11) we deduce

$$b \in Y_p^{(x,b)} \subseteq Z_{k+p}^{(x,a)} \tag{14}$$

Based on (14) and the definition of $Y_0^{(x,b)} = \{b\}$ we can write

$$Y_p^{(x,b)} = Y_0^{(x,b)} \cup U_p^{(x,b)} = \{b\} \cup U_p^{(x,b)} \tag{15}$$

where $U_p^{(x,b)} = Y_p^{(x,b)} \setminus \{b\}$.
If we define the sequence $\{U_{n+1}^{(x,b)}\}_{n\geq p}$ by the relation

$$U_{n+1}^{(x,b)} = \bigcup_{z\in U_n^{(x,b)}} F(x,z) \tag{16}$$

then we observe that based on (12) we obtain

$$Y_{p+1}^{(x,b)} = \bigcup_{t\in Y_p^{(x,b)}} F(x,t)$$

and using (15) and (16) we have

$$Y_{p+1}^{(x,b)} = F(x,b) \cup \bigcup_{z\in U_p^{(x,b)}} F(x,z) = Y_1^{(x,b)} \cup U_{p+1}^{(x,b)}$$

therefore

$$Y_{p+1}^{(x,b)} = Y_1^{(x,b)} \cup U_{p+1}^{(x,b)} \tag{17}$$

We observe that

$$Y_{p+2}^{(x,b)} = \bigcup_{z\in Y_{p+1}^{(x,b)}} F(x,z) = \bigcup_{z\in Y_1^{(x,b)}} F(x,z) \cup \bigcup_{z\in U_{p+1}^{(x,b)}} F(x,z) = Y_2^{(x,b)} \cup U_{p+2}^{(x,b)}$$

therefore

$$Y_{p+2}^{(x,b)} = Y_2^{(x,b)} \cup U_{p+2}^{(x,b)} \tag{18}$$

By induction on $r \geq 0$ we verify now that

$$Y_{p+r}^{(x,b)} = Y_r^{(x,b)} \cup U_{p+r}^{(x,b)} \tag{19}$$

The relation (19) is true for $r \in \{0,1,2\}$ as we have seen in (15), (17) and (18). Suppose that (19) is true for $r = s$. Using (17) we obtain

$$Y_{p+s+1}^{(x,b)} = \bigcup_{z\in Y_{p+s}^{(x,b)}} F(x,z) = \bigcup_{z\in Y_s^{(x,b)}} F(x,z) \cup \bigcup_{z\in U_{p+s}^{(x,b)}} F(x,z) = Y_{s+1}^{(x,b)} \cup U_{p+s+1}^{(x,b)}$$

and thus (19) is true for $r = s+1$.
From (19) we obtain

$$Y_r^{(x,b)} \subseteq Y_{p+r}^{(x,b)} \tag{20}$$

for every $r \geq 0$. Applying (20) for $r = p, 2p, \ldots$, we obtain

$$Y_p^{(x,b)} \subseteq Y_{2p}^{(x,b)} \subseteq \ldots \subseteq Y_{mp}^{(x,b)} \subseteq \ldots$$

for every natural number $m \geq 1$. From $b \in Y_p^{(x,b)}$ and (11) we deduce that $b \in Y_{mp}^{(x,b)}$ for every $m \geq 1$. Using (11) we obtain $b \in Z_{k+mp}^{(x,a)}$ for every $m \geq 1$ therefore $Z_{k+mp}^{(x,b)} \neq \emptyset$ for every $m \geq 1$. Applying Proposition 7.2 we obtain $Z_n^{(x,b)} \neq \emptyset$ for every $n \geq 0$. Thus the proposition is proved. ■

Proposition 7.4. *If $Z_n^{(x,a)} \neq \emptyset$ for every $n \geq 0$ then there is $b \in \bigcup_{n\geq 0} Z_n^{(x,a)}$ such that $b \in [b]_x$.*

Proof. Suppose that $Z_n^{(x,a)} \neq \emptyset$ for every $n \geq 0$. By an *a-chain* we understand a finite or infinite sequence $(a, y_1, y_2, \ldots)$ such that $y_1 \in F(x,a)$ and $y_{i+1} \in F(x,y_i)$ for every $i \geq 1$. Equivalently we can say that an a-chain $(a, y_1, y_2, \ldots)$ is given by a sequence of derivation

$$a \Rightarrow_x y_1 \Rightarrow_x y_2 \Rightarrow_x \ldots$$

Moreover, if we have such a chain then $y_i \in Z_n^{(x,a)}$. For a finite a-chain $(a, y_1, \ldots, y_s)$ we say that the length of this sequence is s.
We consider the set of all a-chains and we prove now that there is an infinite a-chain. By

contrary, suppose that every a-chain is finite and denote by k the maximal length. Because $Z_n^{(x,a)} \neq \emptyset$ for every $n \geq 0$ it follows that $Z_{k+1}^{(x,a)} \neq \emptyset$ and choose an element $z_{k+1} \in Z_{k+1}^{(x,a)}$. But $Z_{k+1}^{(x,a)} = \bigcup_{t \in Z_k^{(x,a)}} F(x,t)$. It follows that there is $z_k \in Z_k^{(x,a)}$ such that $z_{k+1} \in F(x, z_k)$. We reiterate this property and we find a sequence $(z_1, \dots, z_{k+1})$ such that $z_1 \in Z_1^{(x,a)}$ and $z_{i+1} \in F(x, z_i)$ for every $i \in \{1, \dots, k\}$. But $Z_1^{(x,a)} = F(x,a)$ and $z_1 \in Z_1^{(x,a)}$ therefore $(a, z_1, \dots, z_{k+1})$ is an a-chain. We obtained an a-chain of length $k+1$, which is not possible.
It follows that there is an infinite a-chain $(a, y_1, y_2, \dots)$. Suppose that $Attr(K)$ contains r elements, $r \geq 1$. Denote $y_0 = a$. The sequence y_0, y_1, ..., y_r includes $r+1$ elements from $Attr(K)$. There are $i, j \in \{0, \dots, r\}$ such that $i < j$ and $y_i = y_j$.
By induction on m we verify now that $y_{m+1} \in [y_m]_x$ for $m \in \{0, \dots, r-1\}$. Really, $y_{m+1} \in F(x, y_m)$ and $[y_m]_x = \bigcup_{n \geq 1} V_n^{(x,a)}$, where

$$\begin{cases} V_0^{(x,a)} = \{y_m\} \\ \\ V_{n+1}^{(x,a)} = \bigcup_{t \in V_n^{(x,a)}} F(x,t), \ n \geq 0 \end{cases}$$

It follows that $y_{m+1} \in V_1(x,a)x \subseteq [y_m]_x$. Thus the sequence $y_0, y_1, \dots, y_r$ satisfies the property $y_{m+1} \in [y_m]_x$ for every $m \in \{0, \dots, r-1\}$. By Proposition 7.1 we know that if $z \in [u]_x$ and $u \in [v]_x$ then $z \in [v]_x$. It follows that $y_{i+1} \in [y_i]$, $y_{i+2} \in [y_{i+1}]_x$, ..., $y_j \in [y_{j-1}]_x$, therefore $y_j \in [y_i]_x$. But $y_j = y_i$, $y_i \in Z_n^{(x,a)}$ and $y_i \in [y_i]_x$. The proposition is proved. ■

Proposition 7.5. *The following sentences are equivalent:*
(i) $Z_n^{(x,a)} \neq \emptyset$ for every $n \geq 0$.
(ii) There is $b \in \bigcup_{n \geq 0} Z_n^{(x,a)}$ such that $b \in [b]_x$.

Proof. Immediate from Proposition 7.4 and Proposition 7.5. ■

Proposition 7.6. *For every $n \geq 1$ the following property is satisfied: if there is $c_n \in Z_n^{(x,a_1)}$ such that $Val_{attr}(x, c_n) = error$ then there is $c_{n-1} \in Z_{n-1}^{(x,a_1)}$ such that $Val_{attr}(x, c_{n-1}) = error$.*

Proof. If $c_n \in Z_n^{(x,a_1)}$ then there is $t \in Z_{n-1}^{(x,a_1)}$ such that $t \Rightarrow_x c_n$. The value $Val_{attr}(x,t)$ is the value returned by some procedure p such that the procedure call contains the actual argument $Val_{attr}(x, c_n)$. This value is *error*, therefore by the assumptions described in the last part of Section 5 we obtain $Val_{attr}(x,t) = error$. Take $c_{n-1} = t$ and the proposition is proved. ■

Proposition 7.7. *If $c \in Z_i^{(x,a_1)}$ and $Val_{attr}(x,c) = error$ then $Z_1^{(x,c)} \neq \emptyset$ and there is $d \in Z_1^{(x,c)}$ such that $Val_{attr}(x,d) = error$.*

Proof. Suppose by contrary that $Z_1^{(x,c)} = \emptyset$. From the definition of $Z_1^{(x,c)}$ we deduce that $(x,c) \notin C_K(proc)$. Applying Remark 7.1 we deduce that $Val_{attr}(x,c) \in (V_{dir} \times Param) \cup \{unknown\}$, which is not true. Thus the assumption $Z_1^{(x,c)} = \emptyset$ is false. Let us prove that there is $d \in Z_1^{(x,c)}$ such that $Val_{attr}(x,d) = error$. By contrary we suppose that for every $d \in Z_1^{(x,c)}$ we have $Val_{attr}(x,d) \in (V_{dir} \times Param) \cup \{unknown\}$. The value $Val_{attr}(x,c)$ is the value returned by a procedure call whose actual arguments are the values $Val_{attr}(x,d)$ for all $d \in Z_1^{(x,c)}$. These values belong to $(V_{dir} \times Param) \cup \{unknown\}$ and thus the corresponding procedure returns a value from $(V_{dir} \times Param) \cup \{unknown\}$. In other words $Val_{attr}(x,c) \in (V_{dir} \times Param) \cup \{unknown\}$, which is not true. This shows that the property $Val_{attr}(x,d) \in (V_{dir} \times Param) \cup \{unknown\}$ for all $d \in Z_1^{(x,c)}$ is false and the proposition is proved. ■

Proposition 7.8. *If $c \in Z_i^{(x,a_1)}$ and $Val_{attr}(x,c) = error$ then $Z_{i+1}^{(x,a_1)} \neq \emptyset$ and there is $d \in Z_{i+1}^{(x,a_1)}$ such that $Val_{attr}(x,d) = error$.*

Proof. Really, $Z_1^{(x,c)} \subseteq Z_{i+1}^{(x,a_1)}$ and the proposition is obtained from Proposition 7.7. ■

Corollary 7.1. *If $Val_{attr}(x, a_1) = error$ then $Z_n^{(x,a_1)} \neq \emptyset$ for ever $n \geq 0$.*

Proof. We have $a_1 \in Z_0^{(x,a_1)}$ and $Val_{attr}(x, a_1) = error$. We apply now Proposition 7.8 for $i = 0, 1, \ldots$. ■

Proposition 7.9. *$Val_{attr}(x, a_1) = error$ if and only if there is $b \in \{a_1\} \cup [a_1]_x$ such that $b \in [b]_x$.*

Proof. Suppose that there is $b \in \{a_1\} \cup [a_1]_x$ such that $b \in [b]_x$. In order to compute $Val_{attr}(x, b)$ we use $Val_{attr}(x, c)$ for every $c \in [b]_x$. In particular the value $Val_{attr}(x, b)$ is used because $b \in [b]_x$. By Definition 6.1 we deduce that $Val_{attr}(x, b) = error$ because $Val_{attr}(x, b)$ can not be computed. It remains to prove that $Val_{attr}(x, a_1) = error$.We have two cases:
(1) If $b = a_1$ then $Val_{attr}(x, a_1) = error$ because $Val_{attr}(x, b) = error$.
(2) If $b \in [a_1]_x$ then in order to compute $Val_{attr}(x, a_1)$ all values $Val_{attr}(x, c)$ for $c \in [a_1]_x$ are used. Particularly we use the value $Val_{attr}(x, b) = error$. But $b \in [a_1]_x = \bigcup_{n\geq 1} Z_n^{(x,a_1)}$ and thus there is $n \geq 1$ such that $b \in Z_n^{(x,a_1)}$. We apply Proposition 7.6 and we deduce that there is $c_0 \in Z_0^{(x,a_1)}$ such that $Val_{attr}(x, c_0) = error$. But $Z_0^{(x,a_1)} = \{a_1\}$ therefore $c_0 = a_1$. It follows that $Val_{attr}(x, a_1) = error$.

Suppose now that $Val_{attr}(x, a_1) = error$. From Corollary 7.1 we have $Z_n^{(x,a_1)} \neq \emptyset$. We apply Proposition 7.5 and we deduce that there is $b \in \bigcup_{n\geq 0} Z_n^{(x,a_1)}$ such that $b \in [b]_x$. But $\bigcup_{n\geq 0} Z_n^{(x,a_1)} = \{a_1\} \cup [a_1]_x$ and thus the proposition is proved. ■

8. A problem of decidability

Decidability is a term appeared first time in mathematical logic, automata theory and formal languages ([3]). A problem is decidable if there exists an effective procedure, i.e., an algorithm which solves the problem. If no such procedure exists, then the problem is undecidable. As a general reference to decidability and undecidability can be obtained from [7]. Basic undecidable problems include: the halting problem for Turing machines, the Post Correspondence Problem ([6]) and Hilbert's Tenth problem. In the same domain of undecidability we mention the equivalence problem of integer weighted finite automata ([5]).

We studied in the previous section several computational properties of the mapping Val_{attr}. In this section we study the following problem: *decide whether or not $Val_{attr}(x, a_1) = error$*. As we have seen in the previous section $Val_{attr}(x, a_1) = error$ if and only if the computation of $Val_{attr}(x, a_1)$ requires an infinite number of steps. In what follows by the *error-problem* we understand the decision YES or NO in the computation of the value *error* for Val_{attr}. Equivalently this problem can be stated as follows: decide whether or not the value $Val_{attr}(x, a)$ can be computed in a finite number of steps.

Proposition 8.1. *The error-problem for V_{attr} is decidable.*

Proof. In other words we have to prove that there is an algorithm to decide if $Val_{attr}(x, a_1)$ can be computed in a finite many steps. The following algorithm is based on the results treated in the previous section. The algorithm gives the output YES if the computations requires a finite number of steps and NO in the other case.

Error-problem algorithm

```
Input: an arbitrary element $(x, a_1) \in Obj(X) \times Attr(K)$
If $(x, a_1) \notin C_K(proc)$ then output YES.
If $(x, a_1) \in C_K(proc)$ then
    - If there is $b \in \{a_1\} \cup [a_1]_x$ such that $b \in [b]_x$ then output NO
```

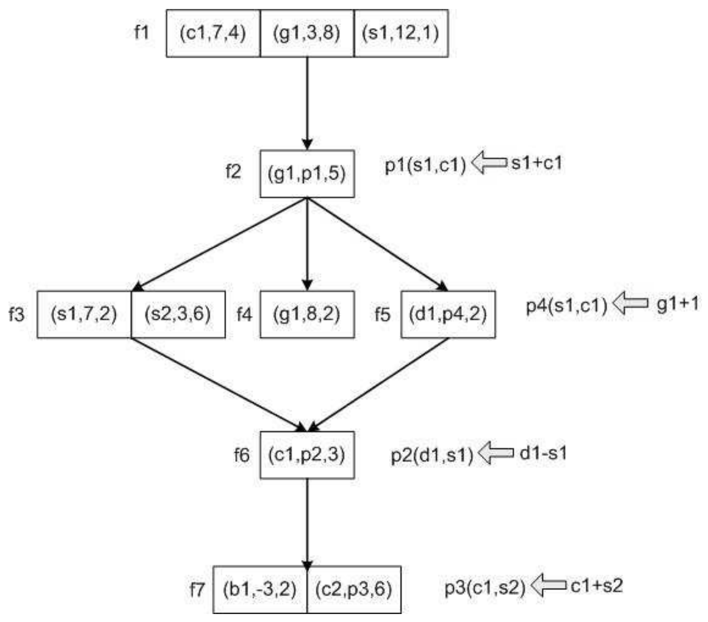

FIGURE 7. An example of knowledge base

- Otherwise output YES

End of algorithm.

Remark 8.1. *In order to finalize our conclusion concerning the termination of this algorithm we observe that for every $a \in Attr(K)$ the set $[a]_x$ is a finite set because $[a]_x \subseteq Attr(K)$ and $Attr(K)$ is finite.*

Example 8.1. *Let us consider the inheritance knowledge base K represented in Figure 7. We relieve the following aspects:*

- *$Obj(K) = \{f1, \ldots, f7\}$;*
- *$Proc(K) = \{p1, p2, p3, p4\}$;*
- *Param is the set of all natural numbers;*
- *$Arg(p1) = (s1, c1)$, $Arg(p2) = (d1, s1)$, $Arg(p3) = (c1, s2)$, $Arg(p4) = (g1)$*

We obtain the following computations:

- *$F(f7, c2) = \{c1\}$; $F(f7, c1) = \{d1\}$; $F(f7, d1) = \{g1\}$; $F(f7, g1) = \{c1\}$;*
- *$c2 \Rightarrow_{f7} c1$; $c1 \Rightarrow_{f7} d1$; $d1 \Rightarrow_{f7} g1$; $g1 \Rightarrow_{f7} c1$;*
- *$Z_0^{(f7,c_2)} = \{c2\}$; $Z_1^{(f7,c_2)} = \{c1\}$; $Z_2^{(f7,c_2)} = \{d1\}$; $Z_3^{(f7,c_2)} = \{g1\}$; $Z_4^{(f7,c_2)} = \{c1\} = Z_1^{(f7,c_2)}$;*
- *$Z_n^{(f7,c_2)} \neq \emptyset$ for every $n \geq 0$*
- *$[c2]_{f7} = \bigcup_{n \geq 1} Z_n^{(f7,c_2)} = \{c1, d1, g1\}$*
- *$[c1]_{f7} = \{d1, g1, c1\}$*

It follows that $Val_{attr}(f7, c2) = error$ because $c1 \in [c2]_{f7}$ and $c1 \in [c1]_{f7}$.

9. The classical case

The classical case of the inheritance representation does not contain the use of parameters as in the present paper. As a consequence, the classical representation includes several features in comparison with the representation presented in this paper:

- $Val_{attr}(x, a) \in V_{dir} \cup \{unknown, error\}$
- A slot is an element of the set $L_{attr} \times (V_{dir} \cup L_{proc})$.
- If (a, v_1) and (b, v_2) are slots for some object then $a \neq b$.
- For every pair $(x, a) \in Obj(K) \times Attr(K)$ there is at most one parent of x that contains the attribute a. It follows that the set $Attr^s_K(x, a)$ contains at most one element.
- The mapping $Choice$ is not used.
- The mapping adj is not used because the set $Param$ is not used.
- The mapping Val_{attr} becomes the following function:

$$Val_{attr} : Obj(K) \times Attr(K) \longrightarrow V_{dir} \cup \{unknown, error\}$$

  - If $h(x, a_1) = v_1$ and $v_1 \in V_{dir}$ then $Val_{attr}(x, a_1) = v_1$.
  - If $h(x, a_1) = p_1$ and $p_1 \in Proc(K)$ then we consider the following two cases:
    (1) If $Arg(p_1) = (b_1, \ldots, b_r)$ and $Val_{attr}(x, b_1)$, ..., $Val_{attr}(x, b_r)$ are elements of V_{dir} then $Val_{attr}(x, a_1) = p_1(Val_{attr}(x, b_1), \ldots, Val_{attr}(x, b_r))$
    (2) Otherwise $Val_{attr}(x, a_1) = error$.
  - If $h(x, a_1) = no$ then $Val_{attr}(x, a_1) = unknown$.

10. Conclusions and future work

In this paper we introduced an extension of the classical knowledge bases which use the inheritance mechanism. The multiple inheritance is allowed an every attribute value can specify a parameter. Special features of an attribute can be defined by these parameters: the quality of a product, the uncertainty of the attribute, the risk etc. The initial value of an attribute includes parameters and during the valuation process some rule of combination can be used for the case when several attribute values are inherited. The interrogation of such a knowledge base is defined and the computational properties of the answer function are studied. This function is a recursive one. An necessary and sufficient condition for the detection the case of an infinite number of steps in the valuation process of this function is given.

The parameter assigned to an attribute value can specify some feature of this value. We can relieve the following particular cases for these parameters:

- **Fuzzy case.**
  A certainty factor is neither a probability nor a truth value. A certainty factor is used to express how accurate, truthful, or reliable a person judges an attribute to be. In this case each p_i defines the certainty factor of the knowledge (a_i, v_i). In other words, the value p_i is the certainty factor for the attribute a_i to have value v_i. We consider that a certainty factor is a value from the real interval $[a, b]$.
- **Probabilistic case.** The conditions imposed in this case can be written as follows:
  - $p \in [0, 1]$ for every $p \in Param$
  - If $s = ord_x(a)$ and $W(x, a) = \{p \in Param \mid \exists (v, p) \in Attr^s_K(x, a)\}$ then for every $(x, a) \in Obj(K) \times Attr(K)$

$$\sum_{p \in W(x,a)} p = 1$$

- **Risk case.**
  In this case the values (a_i, p_i, q_i) specifies the risk factor q_i to choose the value p_i for attribute a_i. Various strategy can be chosen: minimum risk, maximum risk or a combination min-max is also possible depending on the application.

A database can be thought as a particular knowledge base. Hence many subjects treated for knowledge bases are extensions of the corresponding problems for databases. The concepts and the results presented in this paper were suggested by similar problems treated in the domain of data bases. For example, an interesting problem for data bases is developed in [2]: the communication of a client program with a database server through a query language. Starting with the problem of communication between two computers to interrogate a data base we treat in a forthcoming paper the problem of decomposability of an inheritance knowledge base into several disjoint components such that each component can be uploaded on a work station and the computations can be locally performed. This means that if an object belongs to some component then the answer mapping can be computed only be the knowledge contained in the corresponding component. We intend to imply the use of the mobile agents for this computation. More precisely, we are interested to study the decomposition of an inheritance knowledge base into several components such that:

- each component allows to compute the attribute values of its objects such that no other component is used in this process; two distinct component are disjoint;
- each component can be uploaded on a work station of a computer network, where the computations are performed by an agent which is a slave agent;
- a master agent controls ([1]) the whole computation, is connected by the slave agents and communicates with these entities to satisfy the interrogation of the knowledge base.

We are interested also to model the distributive computations based on the inheritance mechanism.

References

[1] **Baumann Joachim** - Mobile Agents: Control Algorithms, Lecture Notes in Computer Science 1658, Springer, 2000

[2] **Bielecki M., Bussche J.V.** - Database Interrogation Using Conjunctive Queries, Lecture Notes in Computer Science, Volume 2572, 259-269, 2002

[3] **Davis Martin** - Computability and Unsolvability, McGraw-Hill Book Company, Inc, New York, 1958

[4] **Francesco Buccafurri, Wolfgang Faber, Nicola Leone** - Disjunctive logic programs with inheritance, Procs. of ICLP-99, MIT Press, 79-93, 1999

[5] **V. Halava and T. Harju** - Undecidability in Integer Weighted Finite Automata, Fund. Inf. 34 189-200, 1999.

[6] **T. Harju and J. Karhumki** - Morphisms. In: G. Rozenberg and A. Salomaa (eds.), Handbook of Formal Languages Vol. 1, Springer, 439-510, 1997.

[7] **G. Rozenberg and A. Salomaa** - Cornerstones of Undecidability, Prentice Hall, 1994.

[8] **S.Rudeanu** - Lattice Functions and Equations, Springer, Discrete Mathematics and Theoretical Computer Science, 2001

[9] **Leora Morgenstern** - Inheritance comes of age: Applying non monotonic techniques to problems in industry, Artificial Intelligence, 103, 237-271, 1998

[10] **N. Ţăndăreanu** - Lattices of Labelled Ordered Trees (I), Annals of the University of Craiova, Mathematics and Computer Science Series, Vol. XXVIII, 29-39, 2001

[11] **N. Ţăndăreanu** - Lattices of Labelled Ordered Trees (II), Annals of the University of Craiova, Mathematics and Computer Science Series, Vol. Vol. XXIX, 137-144, 2002

[12] **N. Ţăndăreanu** - Inheritance-based knowledge systems and their answer functions computation using lattice theory, Romanian Journal of Information Science and Technology, Vol.6, Numbers 1-2, 227-248, 2003

[13] **N. Ţăndăreanu** - Communication by Voice to Interrogate an Inheritance Based Knowledge System, Research Notes in Artificial Intelligence and Digital Communications, Vol.107, 7th International Conference on Artificial Intelligence and Digital Communications, p.1-15, 2007

[14] **Guizhen Yang and Michael Kifer** - Well-Founded Optimism: Inheritance in Frame-Based Knowledge Bases, Lecture Notes in Computer Science, in On the Move to Meaningful Internet Systems, Vol. 2519, ISBN 978-3-540-00106-5, 1013-1032, 2008

(Claudiu Popirlan) Department of Informatics, University of Craiova,
Al.I. Cuza Street, No. 13, Craiova RO-200585, Romania, Tel. & Fax: 40-251412673
E-mail address: popirlan@gmail.com

(Nicolae Ţăndăreanu) Department of Informatics, University of Craiova,
Al.I. Cuza Street, No. 13, Craiova RO-200585, Romania, Tel. & Fax: 40-251412673
E-mail address: `ntand@rdslink.ro`