Journal of Universal Computer Science, vol. 11, no. 2 (2005), 285-302
submitted: 31/7/04, accepted: 15/12/04, appeared: 28/2/05 © *J.UCS*

Processing Inconsistency of Knowledge on Semantic Level

Ngoc Thanh Nguyen
(Institute of Control and Systems Engineering,
Wroclaw University of Technology, Poland
thanh@pwr.wroc.pl)

Abstract: Inconsistency of knowledge may appear in many situations, especially in distributed environments in which autonomous programs operate. Inconsistency may lead to conflicts, for which the resolution is necessary for correct functioning of an intelligent system. Inconsistency of knowledge in general means a situation in which some autonomous programs (like agents) generate different versions (or states) of knowledge on the same subject referring to a real world. In this paper we propose two logical structures for representing inconsistent knowledge: conjunction and disjunction. For each of them we define the semantics and formulate the consensus problem, the solution of which would resolve the inconsistency. Next, we work out algorithms for consensus determination. Consensus methodology has been proved to be useful in solving conflicts and should be also effective for knowledge inconsistency resolution.

Keywords: Inconsistent knowledge, conflicts, consensus methods
Categories: E.1, H.2.1, I.2.4, I.2.11

1 Introduction

In many practical situations in order to solve a problem one often has to gather knowledge from different resources for realizing the task. Nowadays owing to modern computer technologies gathering knowledge is not a hard task at all, but there may be two features of this knowledge, which often cause the decision making process difficult. The first feature is related to the big amount of knowledge, which on one hand contains many useful elements, but on the other hand it often contains also a lot of useless elements. For this problem many methods for information filtering and ordering have been proposed. The second refers to the consistency of the gathered knowledge. Some elements of this knowledge may refer to the same subject, but they are not coherent. Hunter [Hunter, 98] describes the resources of inconsistent knowledge as situations in which one obtains "too much" information.

Generally, a problem of knowledge inconsistency may be formulated as follows: For a given set of logic formulas, one should find out if these formulas are inconsistent or not. As stated in [Nguyen, 04a] inconsistent knowledge can be considered on two levels: *syntactic level* and *semantic level.* On the syntactic level inconsistent knowledge as a set of logic formulae is assumed to have no model [Kifer, 92], that is on their basis one can conclude *false*. For example, for the set of formulae $\{\neg\alpha\vee\beta, \alpha, \neg\beta, \alpha\vee\neg\beta\}$ with the inference engine of standard logic one can easily notice that no

model exists, because in any interpretation if α and $\neg\beta$ are true then $\neg\alpha\vee\beta$ may not be true. On the semantic level, on other hand, considered formulae are related to a real world and on its basis if a fact and its negation may be inferred simultaneously then we say that the inconsistency exists.

On the syntactic level there are known two main approaches for inconsistency leaving: The first relies on knowledge base revision and the second is based on paraconsistent logics. The first approach, most often used in deductive databases, removes data from the base to produce a new consistent database [de Kleer, 86], [Doyle, 79], [Gardenfors, 88]. The disadvantage of this approach is that it is hard to localize the inconsistency and perform the optimal selection, that is with minimal loss of useful information. In the second approach paraconsistent logics are defined. These logics give sensible inferences from inconsistent information. Hunter [Hunter, 98] has defined the following logics: Weakly-negative logic which uses the full classical language; four-valued logic which uses a subset of the classical language; quasi-logical logic which uses the full classical language and enables effectively rewriting data and queries to a conjunction normal form; and finally argumentative logic which reasons with consistent subsets of classical formulae. Hunter has stated that paraconsistent logics can localize inconsistency and their conclusions are sensible with respect to the data. In these logics the inconsistency is not needed to be resolved. However, this kind of logics does not offer strategies for acting on inconsistency and the means for reasoning in presence of inconsistency in existing logical systems are still scant. Apart from the approaches mentioned above, there are a number of attempts to resolving inconsistent data in databases using labeling mechanisms [Balzer, 91], [Fehrer, 93]. Fuzzy logic is also useful in inconsistency leaving for software development [Marcelloni, 01].

On the semantic level a formulae set is considered referring to a concrete real world. Inconsistency arises if in this world a fact and its negation may be inferred. Up to now there are known also two approaches for leaving inconsistency on this level. The first is related to Boolean reasoning and the second concerns measuring up the inconsistency level. In Boolean reasoning [Brown, 90], [Pawlak, 91] an inconsistency resolution task is formulated as an optimisation problem, which is next transformed to such a form of Boolean formula that the first implicant is the solution of the problem. Finally, in the second approach for solving inconsistency its level is determined by means of some consistency function. For this aim Hunter [Hunter, 03] proposes a measure for inconsistency in a set of formulae; Knight [Knight, 02] measures up the inconsistency in a set of sentences; Nguyen and Malowiecki [Nguyen, 04b] define consistency functions which enable calculating the consistency level for so called conflict profiles.

In practice inconsistency of knowledge arises mainly in 3 cases:

- In the first case knowledge is gathered in a period of time and its "topicality" depends on the timestamps [Ferber, 99], [Katarzyniak, 00], [Tessier, 01]. Thus some pieces of knowledge may be inconsistent with some others. For this kind of inconsistency one can actualize the knowledge base by removing from it the non-actual information, thus the remaining information should be consistent. Another approach may use paraconsistent logics because owing to the localization of

inconsistency one can find out which pieces of knowledge are non-actual [Naqvi, 90].

- In the second case, knowledge results from extracting in a database, for example by data mining methods. Extracted rules are dependent on the data and some of them may be contradictory with each other. Here the methods for knowledge base revision could be useful. However, as mentioned above, using these methods may cause the loss of useful rules.
- The third case refers to conflicts of knowledge in distributed environments. For the same subject (conflict issue) the sites may generate different versions (conflict content) of knowledge [Pawlak, 98], [Nguyen, 02b]. For example, agents in a multiagent system may have different opinions on some matter. Thus they are in conflict of knowledge. It seems that for this kind of inconsistency neither knowledge base revision methods nor paraconsistent logics are useful because the former cause loss of information and the later enable to localize the inconsistency but it is not the most important here. The most important is determining a version of knowledge, which is needed for further processing. Boolean reasoning could be useful but it is not practical because for using it the values of conflict features have to be atomic.

In this paper we deal with the inconsistency of knowledge, which arises in the mentioned above third case. We will resolve it on semantic level using consensus methods. For this aim we consider two structures of the logic formulas representing the conflict content: conjunction and disjunction structures. These structures have been first defined and provisionally analyzed in work [Nguyen, 04a]. In this paper we present some their modifications and the deeper analysis.

In earlier works [Nguyen, 02a], [Nguyen, 02b] consensus methods have been proposed for relational structure. In these works a methodology for consensus choice and its applications in solving conflicts in distributed systems is presented. In this methodology consensus problem has been considered on 2 levels. On the first level general consensus methods, which may effectively serve to solving multi-valued and multi-attribute conflicts are worked out. For this aim a consensus system, which enables describing multi-valued and multi-attribute conflicts is defined and analyzed. Next the structures of tuples representing the contents of conflicts are defined as distance functions between these tuples. Two distance functions (ρ and δ) have been defined. Finally the consensus and the postulates for its choice are defined and analyzed. For defined structures algorithms for consensus determination have been worked out. Besides the problems connected with the susceptibility to consensus and the possibility of consensus modification, have been also investigated. The second level concern varied applications of consensus methods in solving of different kinds of conflicts, which often take place in distributed systems. The following conflict solutions are presented: reconciling inconsistent temporal data; solving conflicts of the states of agents' knowledge about the same real world; determining the representation of expert information; creating an uniform version of a faulty situation in a distributed system; resolving the consistency of replicated data and determining optimal interface for user interaction in universal access systems. A multiagent system (named AGWI) aiding information retrieval and reconciling in the Web was also implemented by means of platform IBM Aglets [Nguyen, 05]. Consensus methods are useful in many

applications, for example, in designing and implementation of intelligent user interfaces [Sobecki, 04].

This paper is organized as follows: In [Section 2] we define the logical structures of inconsistent knowledge and their semantics. We use these structures to represent two kinds of knowledge: positive and negative. [Section 3] presents the consensus approach for solving inconsistency for these structures, in which the algorithms for consensus determining are worked out. Some conclusions and directions for future works are included in [Section 4].

2 Logical Structures of Inconsistent Knowledge

2.1 Basic Notions

For representing inconsistent knowledge an approach based on using relational structures has been presented [Nguyen, 02a]. In this paper we present the logical approach for representing this kind of knowledge. On its basis two kind of knowledge (positive and negative) can be represented.

Formally, we assume that a real world is described by means of a finite set A of attributes and a set V of attributes' *elementary values*, where $V=\bigcup_{a\in A}V_a$ (V_a is caled the super domain of attribute a). In short, pair (A,V) is call a real world. Let $\Pi(V_a)$ denote the set of subsets of set V_a and be called the *domain* of attribute a. Let $\Pi(V_B) = \bigcup_{b\in B}\Pi(V_b)$. We accept the following assumption: For each attribute a its value is a set of elementary values from V_a, thus it is an element of set $\Pi(V_a)$. By an elementary value we mean a value, which is not divisible in the system. Thus it is a relative notion, for example, one can assume the following values to be elementary: time units, set of numbers, partitions of a set etc.

We define the following notions [Nguyen, 04a]: Let $T\subseteq A$,

- A tuple of type T is a function r: $T\rightarrow\Pi(V_T)$. Instead of $r(t)$ we will write r_t and a tuple of type T will be written as r_T. The set of all tuples of type T is denoted by $TYPE(T)$. A tuple is elementary if all attribute values are empty sets or 1-element sets. The set of elementary tuples of type T is denoted by $E_TYPE(T)$. Empty tuple, whose all values are empty sets, is denoted by symbol ϕ. Partly empty tuple, whose at least one value is empty, is denoted by symbol θ. Indeed, symbol θ represents not one tuple, but a set of tuples. Expression $r\in\theta$ will mean that in tuple r at least one attribute value is empty and expression $r\notin\theta$ will mean that in tuple r all attribute values are not empty.
- A sum of two tuples r and r' of type T is a tuple r'' of type T ($r''=r\cup r'$) such that $r''_t=r_t\cup r'_t$ for each $t\in T$.
- A product of two tuples r and r' of type T is also a tuple r'' of type T ($r''=r\cap r'$) such that $r''_t=r_t\cap r'_t$ for each $t\in T$.
- Let $r\in TYPE(T)$ and $r'\in TYPE(T)$ where $T\subseteq T'$, we say that tuple r is included in tuple r' (that is $r\prec r'$), iff $r_t\subseteq r'_t$ for each $t\in T$.

An expression $(a = v)$ or $(a \neq v)$ where $a\in A$, $v\in\Pi(V_a)$ and v is a finite set, is called a *literal* from real world (A,V) (or (A,V)-based for short). If a literal has form $(a = v)$ we call it a *positive literal* if it has form $(a \neq v)$ then we call a *negative literal.* A

negative literal ($a \neq v$) can be considered to be equivalent to $\neg(a = v)$. Positive literals serve for agents to express their positive knowledge, that is knowledge consisting of statements "something should take place", while negative literals serve to express their negative knowledge, that is knowledge consisting of statements "something should not take place". However, a negative literal may be transformed into a positive literal using the attribute super domains, that is literal ($a \neq v$) is equivalent o literal ($a = v'$) where $v' = V_a \backslash v$. Thus if V_a is a finite set one can operate only positive literals. Notice, however, that this equivalence is not always true in processing agent knowledge. For example, a meteorological agent can believe that it will not be rain between 10 a.m. and 12 a.m., but this does not means that in the opinion of the agent it will be rain during the rest of the day.

Instead of ($a = v$) we will write (a,v) and instead of ($a \neq v$) we will write $\neg(a,v)$.

2.2 Conjunction Structure

The general aim in this approach is to represent the versions of agents' knowledge (or agents' opinions on some matter), which are inconsistent. We use the standard logic to represent the knowledge. As it was stated in earlier works [Nguyen, 01], [Nguyen, 02a], [Nguyen, 02b] each conflict consists of 3 parameters: *conflict body*, *conflict subject* and *conflict content*. As conflict body we understand a set of agents, which operate in the same environment and generate different states of knowledge; conflict subject – a contentious issue for which the agents are disagreed; and conflict content – the agents' opinions. As a *conflict profile* we understand a finite set, which consists of agents' opinions referring to given subject. In this approach an element of a conflict profile is a formula in the form of a conjunction of literals:

$$l_1 \wedge l_2 \wedge ... \wedge l_n$$

where l_1, l_2,..., l_n are (A,V)-based literals. By C_{AV} we denote the set of all conjunctions of (A,V)-based literals.

Example 1. Let A be a set of attributes *Time*, *Direction*, *Wind_Speed* which represent the parameters of wind in a meteorological system. The super domains of the attributes are the following: V_{Type} = {gusty, moderate}; $V_{Direction}$ = {n, w, e, s, n-w, n-e, s-w, s-e}; V_{Wind_Speed} = [0 km/h, 250 km/h] (this interval represents the set of integers not smaller than 0 and not greater than 250). Examples of formulae representing agents' opinions are the following:

(*Type*, {gusty}) $\wedge$ (*Direction,* {n}) $\wedge$ (*Wind_Speed,* {70}),
(*Type*, {gusty}) $\wedge$ (*Direction,* {n-w, n}) $\wedge$ (*Wind_Speed,* [100, 200]),
(*Type*, $\varnothing$) $\wedge$ (*Wind_Speed,* [20, 50]).

The empty value of attributes means that in the opinion of an agent the attribute should not have value. We should now define the semantics of conflict profile's elements. The set of attributes appearing in a disjunction is called its *type*.

Definition 1. *As the semantics of a conjunction we understand the following function*:

$$S_C: C_{AV} \rightarrow \Pi(E_TYPE(B)),$$

such that

$$S_C(x) = \{r \in \bigcup_{T \subseteq B} E_TYPE(T) : r_T \prec x'_T \text{ and } r_a = \emptyset \text{ iff } x'_a = \emptyset \text{ for } a \in T\}$$

where $x = (a_1, v_1) \wedge (a_2, v_2) \wedge ... \wedge (a_k, v_k)$,
$x' = \langle (a_1, v_1), (a_2, v_2), ..., (a_k, v_k) \rangle$,

and
$B = \{a_1, a_2, ..., a_k\}$.

Thus the semantics of conjunction x is the set of all elementary tuples which are included in the tuple x' representing x, and their values referring to an attribute are empty if and only if the value of x' is empty for this attribute. The intuition of this definition is based on the aspect that if conjunction x represents the opinion of an agent for some issue then set $S_C(x)$ consists of all possible scenarios which are included in x and may take place according to the agent's opinion. Example 2 should illustrate the intuition.

Example 2. For the real world defined in Example 1 let the agent's opinion be the following:

$x = (Type, \{gusty\}) \wedge (Direction, \{n\text{-}w, n\}) \wedge (Wind_Speed, \{200\})$.

Then the semantics of x can be represented by the following table:

Type	***Direction***	***Wind_Speed***
gusty	n-w	200
gusty	n	200
	n-w	200
	n	200
gusty		200
gusty	n	
gusty	n-w	
gusty		
	n	
	n-w	
		200

Table 1: The semantics of formula x

Each of elementary tuples from this table represents a scenario, which in the opinion of the agent should take place. Notice that the lack of some attribute values in several tuples follows from the types of these tuples.

The tuple x' in Definition 1 is called *corresponding* to formula x. Now we consider another logical structure.

2.3 Disjunction Structure

This structure has been first time proposed in [Nguyen, 04a]. In this paper we provide its deeper analysis. Notice that in the previous section we assume that a negative literal can be transformed into a positive literal if the super domain of the attribute is finite. This assumption is needed in practice because otherwise the semantics of a

conjunction containing this literal should be infinite. This means that the agent's opinion represented by this formula is too imprecise.

We consider now another form of agent knowledge, in which an agent expresses its knowledge in the form of a disjunction of literals. We assume also the existence of positive literals as well as negative literals.

In this structure an agent opinion has the following form:

$$l_1 \vee l_2 \vee ... \vee l_m$$

Notice that owing to this structure an agent may express other type of its opinion than in conjunction structure, viz an agent can now give its opinion in form of a disjunction referring to a number of scenario attributes.

Notice that formula $(l_1 \vee l_2 \vee ... \vee l_m)$ can be treated as a clause. We can then write

$$(l_1 \vee l_2 \vee ... \vee l_m) \equiv (h_1 \vee h_2 \vee ... \vee h_k) \vee (\neg b_1 \vee \neg b_2 \vee ... \vee \neg b_l)$$

where h_1, h_2, ..., h_k are positive literals and $\neg b_1$, $\neg b_2$, ..., $\neg b_l$ are negative literals among l_1, l_2, ..., l_m. Further we have

$$\begin{aligned}(l_1 \vee l_2 \vee ... \vee l_m) &\equiv (h_1 \vee h_2 \vee ... \vee h_k) \vee \neg(b_1 \wedge b_2 \wedge ... \wedge b_l) \\ &\equiv (b_1 \wedge b_2 \wedge ... \wedge b_l) \rightarrow (h_1 \vee h_2 \vee ... \vee h_k)\end{aligned}$$

Formula $(b_1 \wedge b_2 \wedge ... \wedge b_l)$ is called a *body* of the clause, and formula $(h_1 \vee h_2 \vee ... \vee h_k)$ is called a *head* of the clause. It is the well-known form of clauses. The mentioned above transformation shows that the disjunction structure should be very useful in practice for agents to express their opinions.

We now define the semantic of bodies and heads of clauses, which is based on some real world. A clause c is based on a real world (A,V) (or (A,V)-based for short) if each its literal is (A,V)-based. Then a body of a (A,V)-based clause as a conjunction should have the following form: $(a_1, v_1) \wedge (a_2, v_2) \wedge ... \wedge (a_k, v_k)$ for $\{a_1,a_2,...,a_k\} \subseteq A$.

By B_{AV} we denote the set of all bodies of (A,V)-based clauses. The semantic of clause bodies is defined as follows:

Definition 2. *The semantic of clause bodies is defined by the following function*:

$$S_B: B_{AV} \rightarrow \Pi(\bigcup_{T \subseteq A} E_TYPE(T)),$$

such that

$$S_B(b) = \{r \in \bigcup_{B \subseteq T \subseteq A} E_TYPE(T) : r_B \prec b', r_a = \emptyset \text{ iff } b'_a = \emptyset \text{ for } a \in B \text{ and } r_{T \setminus B} \notin \theta\}$$

where

$$b = (a_1,v_1) \wedge (a_2,v_2) \wedge ... \wedge (a_k,v_k),$$
$$b' = \langle (a_1,v_1), (a_2,v_2), ..., (a_k,v_k) \rangle,$$

and

$$B = \{a_1,a_2,...,a_k\}.$$

The body expresses the condition (in the opinion of an agent), which, if fulfilled, will cause occurrence of some scenarios (represented in the head of the clause). It follows that a scenario satisfies the condition b of the agent if it belongs to set $S_B(b)$.

Example 3. For the real world defined in Example 1 let the body of agent's opinion be the following:

$$b = (Type, \{\text{gusty}\}) \wedge (Wind_Speed, \{200\}).$$

Then the semantics of b consists of the following tuples:

Type	*Wind_Speed*	*Direction*
gusty	200	n
gusty	200	e
gusty	200	s
gusty	200	w
gusty	200	n-w
gusty	200	n-e
gusty	200	s-e
gusty	200	s-e
gusty	200	

Table 2: The semantics of body b

Each of elementary tuples from this table represents a scenario, which satisfies the condition expressed in the body.

It is not hard to prove the following properties of the semantics of bodies:

Proposition 1. *A body*

$$b* = (a_1, V_{a_1}) \wedge (a_2, V_{a_2}) \wedge ... \wedge (a_n, V_{a_n})$$

where $\{a_1, a_2, ..., a_n\} \subseteq A$ *should have the following semantics*

$$S_B(b*) = \{r \in \bigcup_{T \subseteq A} E_TYPE(T) : r \notin \theta\}.$$

From this property it implies that and any body $b*$ for any $n \leq \text{card}(A)$, should represent the statement "*everything is possible*".

Proposition 2. *Bodies*

$$b = (a_1,v_1) \wedge (a_2,v_2) \wedge ... \wedge (a_k,v_k)$$

and

$$b' = (a_1,v_1) \wedge (a_2,v_2) \wedge ... \wedge (a_k,v_k) \wedge (a,V_a)$$

where attribute a does not occur in b, should have the same semantics, that is

$$S_B(b) = S_B(b').$$

Bodies b and b' having the same semantics are called *equivalent* to each other. Literal (a,V_a) in some sense has the same role as value *true* in the standard logic.

A notice should be made referring to empty values in bodies. If in a body an attribute has empty value then it is understood that the attribute should not appear in the condition. This situation may take place when some attribute expresses a contradictory feature referring to another.

Now we consider the semantics of clause heads. The head of a clause is in the form of a disjunction $(a_1,v_1) \vee (a_2,v_2) \vee ... \vee (a_l,v_l)$. By H_{AV} we denote the set of all heads of (A,V)-based clauses. The semantics of heads is defined as follows:

Definition 3. *The semantics of clause heads is defined by the following function*:

$$S_H: H_{AV} \rightarrow \Pi(E_TYPE(A)),$$

such that

$$S_H(h) = \{r \in E_TYPE(A) : r_H \prec h', r_H = \phi \text{ iff } h' = \phi \text{ and } r_{A \setminus H} = \phi\}$$

for $h = (a_1,v_1) \vee (a_2,v_2) \vee ... \vee (a_l,v_l)$;
$h' = <(a_1,v_1), (a_2,v_2),..., (a_l,v_l)>$

and

$H = \{a_1,a_2,...,a_l\}$.

A head expresses the result (in the opinion of an agent), which should occur if the condition in the body is fulfilled. Its semantics consists of all scenarios, which may take place. In this definition we use similar assumption to well known *closed world assumption*: the disjunction in the clause head is the "most complete" knowledge of an agent on the subject. Therefore, apart from the attributes appearing in the head, other attributes should not have values (that is their values are empty).

Example 4. For the real world defined in Example 1 let the head of agent's opinion be the following:

$h = (Direction, \{n, n\text{-}w, n\text{-}e\}) \vee (Type, \{moderate\})$

Then the semantics of h consists of the following tuples:

Direction	***Wind_Speed***	***Type***
n	∅	moderate
n-w	∅	moderate
n-e	∅	moderate
∅	∅	moderate
n	∅	∅
n-w	∅	∅
n-e	∅	∅

Table 3: The semantics of head h

It is not hard to prove the following:

Proposition 3. *Heads*

$h = (a_1,v_1) \vee (a_2,v_2) \vee ... \vee (a_k,v_k)$

and

$h' = (a_1,v_1) \vee (a_2,v_2) \vee ... \vee (a_k,v_k) \vee (a,\varnothing)$

where attribute a does not occur in h, have the same semantics, that is

$S_H(h) = S_H(h')$.

Heads h and h' having the same semantics are also called *equivalent*. Literal $(a,\varnothing)$ in some sense has the same role as value *false* in the standard logic.

3 Consensus Determination

In this section we present the algorithms for consensus determination for the defined logical structures of conflict profiles. The general consensus problem has been defined in the literature, according to which the consensus for a conflict profile should fulfill the following two conditions: It should at best represent the elements of conflict

profile; and should be the best compromise for the inconsistent opinions of agents. It has been shown [Nguyen, 01] that in general these conditions may not be satisfied simultaneously. The consensus satisfying the first condition should be determined by Kemeny function, which minimizes the sum of distances between consensus and profile elements. The consensus satisfying the second condition, and partially the first condition, should be determined by the function, which minimizes the sum of squared distances between consensus and profile elements.

For the needs of consensus determining for a given structure the distance function should be defined. Next, the criterion for consensus choice should be formulated. In this work we will concern only the consensus choice satisfying the mentioned above conditions. Although there have been defined other criteria [Nguyen, 01], consensus functions satisfying them will be the subject of future work.

3.1 Consensus Determining for Conjunction Structure

3.1.1 Distance Function

Intuitively, the distance between two conjunctions of the same type should be defined on the basis of the difference in their semantics, because the semantics of a conjunction gives its interpretation. Thus firstly we define the difference in semantics of two conjunctions. For $x = l_1 \wedge l_2 \wedge ... \wedge l_n$ and $x' = l_1' \wedge l_2' \wedge ... \wedge l_n'$ and their semantics $S_C(x)$ and $S_C(x')$ we define the difference between two sets of tuples $S_C(x)$ and $S_C(x')$ as the minimal cost needed for transformation of one tuples set into the other. By the operation transforming set $S_C(x)$ into set $S_C(x')$ we mean performing such operations as *adding*, *removing* and *transformation* to the elements of set $S_C(x)$, which in the result give set $S_C(x')$. For the need of the definition of these operations we define the following cost functions:

- Function d_1: $V \rightarrow (0,+\infty)$: specifies the cost for adding (or removing) of an elementary value to (or from) a set.
- Function d_2: $V \times V \rightarrow [0,+\infty)$: specifies the cost for transformation of one elementary value into another.

Similarly like in work [Nguyen, 02a] for functions d_1 and d_2 we accept the following assumptions:

a) Function d_2 is a metric, i.e. for any $x,y,z \in V$ the following conditions are held:
   - $d_2(x,y) \geq 0$, $d_2(x,y) = 0$ if and only if $x=y$,
   - $d_2(x,y) = d_2(y,x)$,
   - $d_2(x,y)+d_2(y,z) \geq d_2(x,z)$;

b) For any $x,y \in V$ $\left|d_1(x) - d_1(y)\right| \leq d_2(x,y) \leq d_1(x)+d_1(y)$.

Condition a) is natural because function d_2 may be treated as a distance function between elements from set V. For condition b) notice that its first inequality is equivalent to the following inequalities

$$d_1(y) \leq d_2(x,y)+d_1(x) \text{ and}$$
$$d_1(x) \leq d_2(x,y)+d_1(y).$$

The first of them corresponds to the following intuition: The optimal way (that is with the minimal cost) for transforming set $\{u\}$ into set $\{u,v\}$ where tuples u and v contain elements x and y respectively ($x,y \in V$), consists of adding element y to set $\{u\}$. The cost needed for this operation is equal $d_1(y)$. Thus this cost should not be greater than the cost of the operation which consists of transforming element x into element y and adding element x. The intuition for the second inequality is similar. The second inequality of condition b) states that the cost of transformation of one elementary value into another should not be greater than the sum of removing cost of the first element and adding cost of the second. The intuition is that if one wants to transform set $\{u\}$ into set $\{v\}$ then generally the cost is not minimal if he removes x from the first set and next adds y.

Let's consider an example:

Example 3. For the real world defined in Example 1 let

$x = (Type=\{\text{gusty}\}) \wedge (Direction=\{\text{n}\}) \wedge (Wind_Speed=\{200\})$, and

$x' = (Type=\{\text{gusty}\}) \wedge (Direction=\{\text{n-w}\}) \wedge (Wind_Speed=\{200\})$

Then the semantics $S_C(x)$ of x is represented by the following table:

Type	***Direction***	***Wind_Speed***
gusty	n	200
	n	200
gusty		200
gusty	n	
gusty		
	n	
		200

Table 4: Set $S_C(x)$

and the semantics $S_C(x')$ of x' is represented by the following table:

Type	***Direction***	***Wind_Speed***
gusty	n-w	200
	n-w	200
gusty		200
gusty	n-w	
gusty		
	n-w	
		200

Table 5: Set $S_C(x')$

As we can see, to transform $S_C(x)$ into $S_C(x')$ it is needed to perform 2 operations: removing value "n" of attribute *Direction* and adding value "n-w" of the same attribute, or only 1 operation: transforming value "n" into "n-w". Because of

inequality $d_2(x,y) \leq d_1(x)+d_1(y)$ the minimal cost should come from the second variant, that is it should be equal d_2(n, n-w).

For the simple calculation way one can assume $d_1(x) = d_1(y) = 1$ and $d_2(x,y) = d_1(x) + d_1(y)$.

Now we can give the definition of the distance between two conjunctions:

Definition 4. *For conjunctions* $x = l_1 \wedge l_2 \wedge ... \wedge l_n$ *and* $x' = l_1' \wedge l_2' \wedge ... \wedge l_n'$ *their distance* $d_C(x,x')$ *is equal the minimal cost for transforming set* $S_C(x)$ *into set* $S_C(x')$.

Notice, however, that it is not convenient to obtain the minimal cost of the transformation $S_C(x)$ into $S_C(x')$ because the semantics of a conjunction is often very large. Therefore, for calculating this distance we will use the distance function δ defined for sets of elementary values given in [Nguyen, 02a]. This definition is very similar to the conception of the difference in semantics of 2 conjunctions. In the same work a distance function φ for tuples has also been defined:

Definition 5. *The distance* $\varphi(r,r')$ *between two tuples r and r' of type B is equal to the following number*

$$\varphi(r,r') = \frac{1}{card(B)} \sum_{a \in B} \delta(r_a, r'_a) .$$

Comparing these definitions it is not hard to prove the following theorem. Let $x = l_1 \wedge l_2 \wedge ... \wedge l_n$ and $x' = l_1' \wedge l_2' \wedge ... \wedge l_n'$, let r and r' be the tuples corresponding to x and x', respectively. Let B be the type of conjunctions x and x' (that is it is also the type of tuples r and r'). Then we have:

Theorem 1. $d_C(x,x') = \frac{1}{card(B)} \sum_{a \in B} \delta(r_a, r'_a) .$

Owing to this theorem the calculation of distance between 2 conjunctions is more easy and effective than calculation of the distance between 2 sets $S_C(x)$ and $S_C(x')$.

3.1.2 Consensus Determination

Having defined the distance between conjunctions we can now define the consensus for conflict profile.

Definition 6. *Let* $P \subseteq C_{AV}$ *be a conflict profile being a set with repetitions, then*

a) By a O_1*-consensus for P we call such conjunction* $c \in C_{AV}$ *that*

$$\sum_{x \in P} d_C(c,x) = \min_{c' \in C_{AV}} \sum_{x \in P} d_C(c',x)$$

b) By a O_2*-consensus for P we call such conjunction* $c \in C_{AV}$ *that*

$$\sum_{x \in P} (d_C(c,x))^2 = \min_{c' \in C_{AV}} \sum_{x \in P} (d_C(c',x))^2$$

The O_1-consensus is very well known and the most often used in practice. However, we introduce also O_2-consensus, which minimizes the sum of squared

distances between the consensus and the elements of the profile. The reason is that this kind of consensus also plays very essential role in conflict solving, this follows from the fact that an O_2-consensus guarantees the maximal degree of compromise between contentious opinions of agents and simultaneously is partially the best representative for these opinions [Nguyen, 01].

On the basis of Theorem 1 for determining both kinds of consensus we can use the algorithms presented in work [Nguyen, 02a] for relational structure. The general algorithm for conjunctions may be formulated as follows:

Algorithm 1: Computing consensus for a set of conjunctions.
Given: Finite set $P \subseteq C_{AV}$ of conjunctions of the same type where $P = \{x_1, x_2,..., x_n\}$.
Result: Consensus for P.

BEGIN
1. Create tuples r_1, r_2,..., r_n corresponding to conjunctions x_1, x_2,..., x_n, respectively;
2. Calculate consensus r for tuples r_1, r_2,..., r_n using an algorithm in [Nguyen, 02];
3. Create conjunction c from tuple r

END.

3.2 Consensus Determining for Disjunction Structure

3.2.1 Distance Function

For opinions in disjunction structure first we propose a method for measuring up the distance between 2 clauses

$$c_1 = u_1^{(1)}, u_2^{(1)}, .., u_m^{(1)} \rightarrow t_1^{(1)}, t_2^{(1)}, ..., t_n^{(1)}$$

and

$$c_2 = u_1^{(2)}, u_2^{(2)}, .., u_{m'}^{(2)} \rightarrow t_1^{(2)}, t_2^{(2)}, ..., t_{n'}^{(2)}.$$

Generally, unlike the distance between conjunctions, the bodies and heads must not have the same type. The distance between 2 clauses may be understood as the sum of the distance between the bodies and the distance between the heads of these clauses. Firstly, we deal with the distance between clause bodies.

As for conjunctions, it is intuitive that the distance between 2 bodies should be equal the minimal cost of translating the semantics of the first body to the semantics of the second body. Thus we have:

Definition 7. *For bodies* $b = u_1 \wedge u_2 \wedge ... \wedge u_n$ *and* $b' = u_1' \wedge u_2' \wedge ... \wedge u_m'$ *their distance* $d_B(b,b')$ *is equal the minimal cost for transforming set* $S_B(b)$ *into set* $S_B(b')$.

On the other hand, the distance between bodies can be calculated using only the attribute values in these bodies. Because the bodies must not have the same attributes, it is necessary to transform them to such forms that have the same type. For doing this we can use the property which follows from Proposition 3. Thus for calculating the distance d_B between 2 bodies b and b' the following procedure should be used:

1. Creating 2 equivalent bodies by adding to each of them new literals with attributes which appear in only one of them, and their values are equal to their super domains;

2. Calculating for each attribute the distance between its values in the bodies using function δ assuming $d_1(x) = d_1(y) = 1$ and $d_2(x,y) = d_1(x)+d_1(y)$ (see [Section 3.1]);
3. The distance between the bodies will be equal to the average of the distances calculated in Step 2.

The basis of Step 1 is relied on Proposition 2. Let $b = u_1 \wedge u_2 \wedge ... \wedge u_n$ and $b' = u_1' \wedge u_2' \wedge ... \wedge u_m'$, let r and r' be tuples corresponding to b and b' respectively after performing Step 1 described above. Let B be the type of tuples r and r'. It is not hard to prove the following:

Theorem 2. $d_B(b,b') = \frac{1}{card(B)} \sum_{a \in B} \delta(r_a, r'_a)$.

Let's consider an example.

Example 4. Let $A=\{a_1, a_2, a_3\}$, $V_{a_1} = \{x,y,z\}$, $V_{a_2} = \{1,2,3\}$, $V_{a_3} = \{+,-,\pm\}$. Consider bodies

$$b_1 = (a_1,\{x,y\}) \wedge (a_3,\{+\})$$

and

$$b_2 = (a_1,\{x\}) \wedge (a_2,\{1,2\}).$$

After completing we have 2 bodies b_1' and b_2' equivalent to bodies b_1 and b_2, respectively:

$$b_1' = (a_1,\{x,y\}) \wedge (a_2,\{1,2,3\}) \wedge (a_3,\{+\})$$

and

$$b_2' = (a_1,\{x\}) \wedge (a_2,\{1,2\}) \wedge (a_3,\{+,-,\pm\}).$$

Using distance function δ we have $\delta(\{x,y\},\{x\}) = 1$; $\delta(\{1,2,3\},\{1,2\}) = 1$; and $\delta(\{+\},\{+,-,\pm\}) = 2$, so the distance $d_B(b_1,b_2)$ between bodies b_1 and b_2 should be (1+1+2)/3 = 4/3.

The distance d_H between 2 heads of clauses may be defined in the similar way. That is:

Definition 7. *For heads* $h = t_1 \wedge t_2 \wedge ... \wedge t_n$ *and* $h' = t_1' \wedge t_2' \wedge ... \wedge t_m'$ *their distance* $d_H(h,h')$ *is equal the minimal cost for transforming set* $S_H(h)$ *into set* $S_H(h')$.

The calculation of distance $d_H(h,h')$ may be performed in the similar way as calculation of $d_B(b,b')$. The difference is based only on the first step. Here one should create 2 equivalent heads by adding to each of them new literals with attributes, which appear in only one of them, and their values are equal empty set.

Example 5. Using the parameters defined in Example 4 let's consider the following heads:

$$h_1 = (a_1,\{x,y\}) \vee (a_3,\{+\})$$

and

$$h_2 = (a_1,\{x\}) \vee (a_2,\{1\}).$$

After completing we obtain 2 heads h_1' and h_2' equivalent to heads h_1 and h_2, respectively:

$$h_1' = (a_1,\{x,y\}) \vee (a_2,\varnothing) \vee (a_3,\{+\})$$

and

$$h_2' = (a_1,\{x\}) \vee (a_2,\{1\}) \vee (a_3,\varnothing).$$

Using distance function δ we have $\delta(\{x,y\},\{x\}) = 1$; $\delta(\varnothing,\{1\}) = 1$; and $\delta(\{+\},\varnothing) = 1$, so the distance $d_H(h,h')$ between heads h and h' should be (1+1+1)/3 = 1.

Now we deal with the calculating distance between 2 clauses. The distance d between 2 clauses

$$c = b \rightarrow h$$

and

$$c = b' \rightarrow h'$$

can be calculated on the basis of the distances between the bodies and the heads, by means of the following definition:

Definition 8. $d(c,c') = 1/2\ d_B(b,b') + 1/2\ d_H(h,h')$,

or more generally

$$d(c,c') = \alpha \cdot d_B(b,b') + \beta \cdot d_H(h,h'),$$

where α *and* β *are parameters representing the weights of distances between bodies and heads*, *such that* $\alpha,\beta \geq 0$ *and* $\alpha+\beta=1$.

Since function δ is a metric, function d is also a metric. Functions d_C, d_B, d_H are also metrics.

3.2.2 Consensus Determination

Now, having defined distance function between clauses we can work out an algorithm for consensus determining for a set of clauses. Similarly like for the conjunction structure the consensus problem for disjunction structure may be formulated as follows:

Let $\boldsymbol{C}_{AV}$ denote the set of all (A,V)-based clauses, and let P be a finite subset with repetitions of set $\boldsymbol{C}_{AV}$. A clause $c^* \in \boldsymbol{C}_{AV}$ is called:

- an O_1-consensus of set P if it satisfies the following condition:

$$\sum_{c \in P} d(c^*,c) = \min_{c \in \boldsymbol{C}_{AV}} \sum_{c' \in P} d(c,c');\ \text{and}$$

- an O_2-consensus of set P if it satisfies the following condition:

$$\sum_{c \in P} (d(c^*,c))^2 = \min_{c \in \boldsymbol{C}_{AV}} \sum_{c' \in P} (d(c,c'))^2.$$

For determining a consensus for a clauses set it is needed to complete the clauses by adding attributes to their bodies and heads so that the same attributes occur in each clause. The rule of attribute completing is given above in distance determination way.

Assuming that the attributes from the real world (A,V) are independent, for determining an O_1-consensus we can adopt the algorithm given in [Nguyen, 02a], in which value of each attribute in the consensus can be calculated in an independent way. We present here 2 algorithms: the first calculates a consensus for an attribute and the second creates the consensus for a set of clauses. These algorithms are presented as follows:

Algorithm 2: Computing O_1-consensus c_b for attribute b occurring in the clauses.

Given: Finite set P (with repetitions) of (A,V)-based clauses, distance function δ between attribute values and an attribute b.

Result: O_1-consensus c_b for attribute b.

BEGIN

1. Create the set:
   $profile(b) = \{v_b: (b,v_b)$ appears in a clause of set $P\}$;
2. Let $X:=\emptyset$; $S_b:=\sum_{y\in profile(b)} \delta(X, y)$;
3. Select from $V_b\backslash X$ an element x such that the sum
   $\sum_{y\in profile(b)} \delta(X \cup \{x\}, y)$ is minimal;
4. If $S_b < \sum_{y\in profile(b)} \delta(X \cup \{x\}, y)$ then
   Begin
   $S_b:=\sum_{y\in profile(e)} \delta(X \cup \{x\}, y)$;
   $X:=X\cup\{x\}$;
   End;
5. If $V_b\backslash(X\cup\{x\}) \neq \emptyset$ then GOTO 3;
6. Let $c_b=X$;

END.

Algorithm 3: Computing O_1-consensus c for set P of clauses.

Given: Finite set P (with repetitions) of (A,V)-based clauses, distance function δ.

Result: O_1-consensus c for P.

BEGIN

1. Complete clauses in P by adding attributes and their values so that all clauses have the same type;
2. For each attribute b calculate the consensus c_b using Algorithm 2;
3. Create the body and the head of consensus c by concatenation of ingredients c_b for b appearing in the bodies and the heads of given clauses, respectively;

END.

For determining an O_2-consensus for set P it is impossible to treat the attributes in an independent way. For many structures of attribute values determining a consensus on the basis of this criterion is a NP-complete problem. An example of such structures is the relational structure presented in [Nguyen, 02b]. In these cases heuristic algorithms should be worked out. Besides, genetic algorithms should also be useful.

4 Conclusions

In this paper two logical structures for representing inconsistent knowledge on semantic level are presented. Both of them are multi-valued and multi-attribute. The first structure is based on representing inconsistent knowledge by conjunctions and the second refers to disjunctions (or clauses). For each structure the semantics and the distance functions are defined. On this basic the consensus problem is formulated and the algorithms for consensus determination are worked out. The future work should concern working out algorithm for determining consensus for according to criterion of minimal sum of squared distances for these structures.

References

[Balzer, 91] R. Balzer, Tolerating Inconsistency. In: Proceedings of the 13th International Conference on Software Engineering. IEEE Press, 1991, 158-165

[Barthelemy, 91] J.P. Barthelemy, M.F. Janowitz, A Formal Theory of Consensus. SIAM Journal of Discrete Mathematics, Vol. 4, 1991, 305-322

[Brown, 90] F.N. Brown, Boolean Reasoning, Kluwer Academic Publishers, Dordrecht, 1991

[Day, 87] W.H.E. Day, Consensus Methods as Tools for Data Analysis. In: H.H. Bock (ed.): *Classification and related Methods of Data Analysis*, Proceedings of IFCS'87, North-Holland, 1987, 317-324

[Doyle, 79] J. Doyle, A Truth Maintenance System, Artificial Intelligence, Vol. 12, 1979, 231-272

[Ferber, 99] J. Ferber, Multi-Agent Systems. Addison Wesley, New York, 1999

[Fehrer, 93] D. Fehrer, A Unifying Framework for Reason Maintenance. In: M. Clark et. al (eds): *Symbolic and Qualitative Approaches to Reasoning and Uncertainty.* LNCS, Vol. 747, 1993, 113-120

[Gardenfors, 88] P. Gardenfors, Knowledge in Flux. MIT Press, 1988

[Hunter, 98] A. Hunter, Paraconsistent Logics. In: D. Gabbay, P. Smets (eds), *Handbook of Defeasible Reasoning and Uncertain Information.* Kluwer Academic Publishers, 1998, 13-43

[Hunter, 03] A. Hunter, Evaluating the Significance of Inconsistencies. In: *Proceedings of the International Joint Conference on AI* (*IJCAI'03*). Morgan Kaufmann, 2003, 468-473

[Katarzyniak, 00] R. Katarzyniak, N.T. Nguyen, Reconciling Inconsistent Profiles of Agents' Knowledge States in Distributed Multiagent Systems Using Consensus Methods. System Science**,** Vol. 26, 2000, 93-119

[Kemeny, 59] J.G. Kemeny, Mathematics without numbers. Daedalus, Vol. 88, 1959, 577–591

[Kifer, 92] M. Kifer, E.L. Lozinski, A Logic for Reasoning with Inconsistency. Journal of Automatic Reasoning, Vol. 9, 1992, 179-215

[De Kleer, 86] J. De Kleer, An Assumption-based TMS. Artificial Intelligence, Vol. 28, 1986, 127-162

[Knight, 02] K. Knight, Measuring Inconsistency. Journal of Philosophical Logic, Vol. 31, 2002, 77-98

[Lipski, 79] W. Lipski, On semantic issues connected with incomplete information databases. ACM Trans. Database Systems, Vol. 4, 1979, 262-269

[Marcelloni, 01] F. Marcelloni, M. Aksit, Leaving inconsistency using fuzzy logic, Information and software technology, Vol. 43, 2001, 725-741

[Naqvi, 90] S. Naqvi, F. Rossi, Reasoning in Inconsistent Databases. In: *Logic Programming*: *Proceedings of the North American Conference.* MIT Press, 1990, 255-272

[Nguyen, 01] N.T. Nguyen, Using Distance Functions to Solve Representation Choice Problems. Fundamenta Informaticae Vol. 48, 2001, 295-314

[Nguyen, 02a] N.T. Nguyen, Consensus System for Solving Conflicts in Distributed Systems. Information Sciences – An International Journal Vol. 147, 2002, 91-122

[Nguyen, 02b] N.T. Nguyen, Methods for Consensus Choice and their Applications in Conflict Resolving in Distributed Systems. Wroclaw University of Technology Press 2002 (in Polish)

[Nguyen, 04a] N.T. Nguyen, Consensus methodology for Inconsistent Knowledge Processing. In: Nguyen N.T. (ed.), *Intelligent Technologies for Inconsistent Knowledge Processing*. Advanced Knowledge International, Adelaide, Australia, 2004, 3-20

[Nguyen, 04b] N.T. Nguyen, M. Malowiecki, Consistency Function for Conflict Profiles". LNCS Transactions on Rough Sets , Vol. 1, 169-186

[Nguyen, 05] N.T. Nguyen, A. Blazowski, M. Malowiecki, A Multiagent System Aiding Information Retrieval in Internet Using Consensus Methods. In: Proceedings of SOFSEM 2005. Lecture Notes in Computer Science Vol. 3381, 2005, 399-402

[Pawlak, 91] Z. Pawlak, Rough Sets - Theoretical Aspects of Reasoning about Data. Kluwer Academic Publishers, 1991

[Pawlak, 98] Z. Pawlak, An Inquiry into Conflicts. Journal of Information Science, Vol. 109, 1998, 65-78

[Sobecki, 04] J. Sobecki, M. Weihberg, Consensus-based Adaptive User Interface Implementation in the Product Promotion; in Keates S. et al. (eds.): "Design for a more inclusive world", Springer-Verlag, 2004, 111-121

[Tessier, 01] C. Tessier, L. Chaudron, H.J. Müller, Conflicting Agents: Conflict Management in Multiagent Systems, Kluwer Academic Publishers, 2001