Automatic Construction of Knowledge-Based System using Knowware System

Sio-Long Lo and Liya Ding
Macau University of Science and Technology
Macau SAR
China

1. Introduction

Knowledge-based system (KBS) is a problem solving approach that makes use of human knowledge in possible ways. Usually, the knowledge used in KBS may be obtained directly from domain expert or through some kind of machine learning based on available data. The quality of knowledge used has an important impact on the performance of KBS. The success of development and application of an intelligent system requires the availability of two groups of people: AI experts who hold the techniques and tools for problem solving, and domain experts who know well the problem to be solved and hold domain knowledge leading to a necessity of the development of intelligent system. However, in reality, it is often a challenge to get the both groups working together to derive the inherent synergies.
Knowware System (KWS) is a framework proposed as development tool for design and development of KBS. KWS offers classes of knowledge-based processing unit to support developer in modelling their KBS, and generates the target KBS based on the definition from developer. A typical KBS generated by KWS is a hybrid intelligent system that contains a knowledge hierarchy and an inference engine. The knowledge hierarchy consisting of multiple components forms a static inference structure in KBS while the inference engine controls the dynamic inference flow through managing execution of components.
The inference in a hybrid KBS constructed by KWS is a truth value flow inference, with knowledge-based processing handled locally in each individual components and a truth value flow throughout the entire KBS. As a uniformed format, interval-valued confidence defined as fuzzy number has been proposed to represent the imprecision and uncertainty during inference. The KWS inference engine realizes control of inference through three aspects: the management of protocol between components, the control of execution order of components, and the confidence transfer.

2. Knowware System

The Knowware System (KWS) has been proposed for the development of knowledge-based systems. It can accept from user knowledge sources represented in varied formats and select appropriate intelligent techniques to construct desired knowledge-based processing units of hybrid KBS, therefore allow the KBS developer more easily and conveniently model and develop a customized intelligent system.

2.1. Hierarchical Modeling of KBS

In a typical application, the mapping relation between inputs and output of the problem may be complex, and description of such a mapping relation using a global knowledge can be difficult and incapacity. A possible strategy is to divide the complex mapping relation to multiple units, with each of the units described by a corresponding local knowledge base, and the type of knowledge and the inference mechanism in each of the units varied upon the specific problem solving and the availability of knowledge. Following this sprit, hierarchical problem representation represents a domain problem with a hierarchy and uses multiple AI techniques for problem solving.

2.2. Construction of KBS using KWS

The hierarchical representation for KBS was introduced by (L. Ding and H.C. Lui, 1999; L. Ding, 2007a). The key idea of hierarchical representation for KBS is hybrid KBS, which consists of multiple sub-KBS constructed in a hierarchical structure (Figure 1). The KWS not only allows developers to easily design their system, but also realizes an automatic construction of the target KBS based on the developers' design.
As a typical development process, KWS receives the description of KBS from developer and then automatically constructs the target KBS. Therefore a Knowledge Description Language (KDL for short) is essential, which will be introduced in Section 2.2.3. Developers can use the KDL text to describe their system, and the text is a kind of input to KWS with a KBS constructed by KWS as the corresponding output. The KBS constructed by KWS is a stand-alone application, so the end-user can use the KBS easily without the care about the details of implementation.

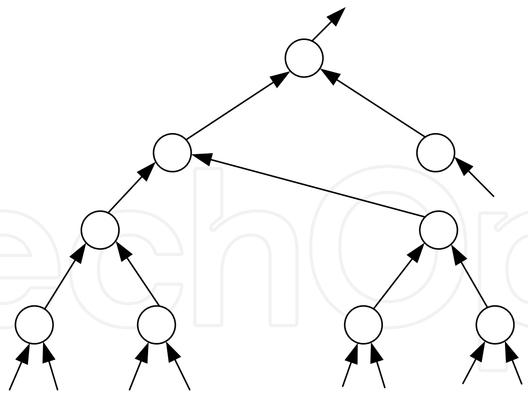


Fig. 1. Structure of KBS constructed by KWS

2.2.1. Sub-Systems of KWS for KBS Construction

There are three subsystems of KWS supporting the automatic construction of customized KBS.
Intelligent Editor – It provides a friendly GUI for the developer to design a KBS. Developers use the graphic description to describe their KBS. Editor also does error checking for the

process of developing KBS. Once design is confirmed, Editor will construct the internal inference structure of target KBS based on the graphic description, and generate the corresponding KDL text.

KDL Processor – It receives the KDL description of a target KBS, and compiles it to a corresponding knowledge hierarchy as the internal inference structure, using suitable intelligent components stored in the warehouse with possible customization. The KDL text can be either from the interactive editor or user's input. In the latter case, it also checks the syntax of KDL text inputted.

Installer – It saves the internal knowledge hierarchy in a suitable data format when a user's definition of target KBS is confirmed. At the last stage, it packs the knowledge hierarchy with the KWS inference engine as well as the installer itself to a stand-alone target application. The embedded installer will be responsible to reload the saved KBS upon user's calling of the application.

2.2.2. Work Flow of KWS

In order to develop a desired intelligent system, the developer can choose any of the knowware that fits into his/her need, via two possible ways. One is to define his/her target system in KDL text and then call the KDL processor for compilation to generate the internal inference structure. The other alternative is to use the intelligent editor to design the target system step-by-step and get the target knowledge hierarchy constructed after confirmation. In the latter case, the editor also generates a corresponding KDL text so the developer can make modification conveniently later on. For a KBS successfully constructed, the installer will save the internal inference structure to a suitable format and reconstruct it later upon request. Figure 2 shows the work flow of KWS.

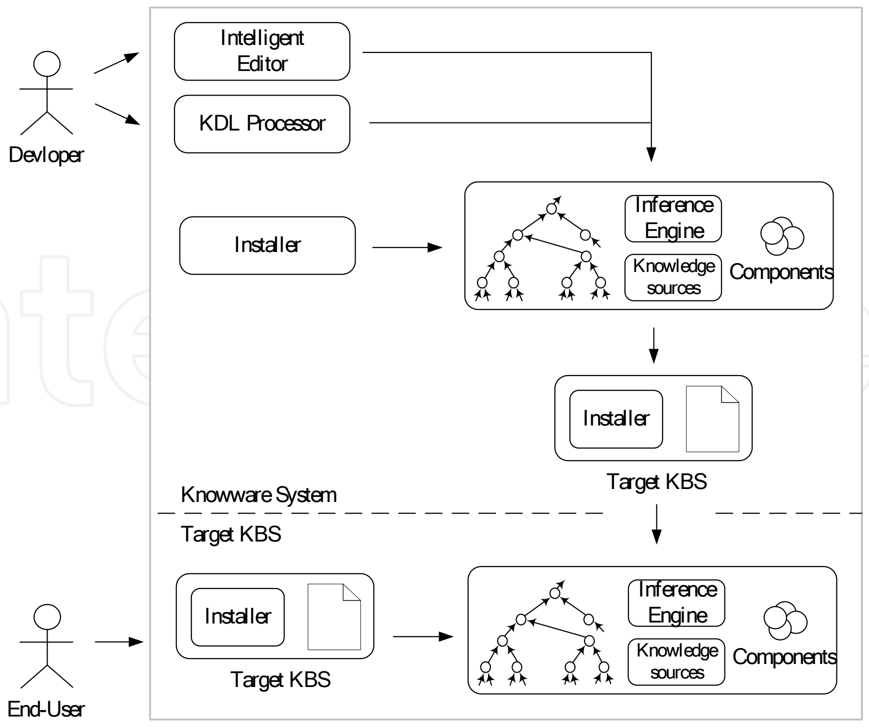


Fig. 2. Work flow of KWS

2.2.3. Knowledge Description Language

The Knowledge Description Language makes it possible for developers to describe their target KBS in a text format. The knowledge-based processing units offered by KWS will be used as building blocks to make up the KBS. The input/output of each intelligent component (IC) called field must be specified, this information indicates the linking between ICs and a pipeline for data connection with ICs. A KDL text consists of the following parts: 1) declaration of fields, each including name and data type; 2) declaration of intelligent components, each including name, component class and type, knowledge source, fields of input(s) and output linked with. The details of intelligent component and field will be presented in Sections 2.3. An example of KDL text is shown in Figure 3.

```
Support-Field-Name = ( Field1 ) , Support-Field-Data { Char ( 2 ) }
Result-Field-Name = ( Field2 ) , Result-Field-Data { Char ( 2 ) }
InCom-Name = ( Filter-01 ) , InCom-Body {
            Filter Dictionary
            NoCondition
            Standard { Program = ( Standard Dictionary ) ,
                   Knowledge-Source = ( Filter01_Knowledge ) }
            Input = ( Field = ( Field1 ) ) , Output = ( Field = ( Field2 ) )
}
```

Fig. 3. An example of KDL

2.2.4. Generation of Target KBS

The last process of developing KBS using KWS is the generation of target KBS. The knowledge hierarchy will be recorded in a data-file. By packing the target KBS with hierarchy record, corresponding components, knowledge sources, inference engine, and installer, the KBS for the end-user is obtained as a stand-alone system. The task for packing and reloading of KBS is done by the Installer which also provides a GUI to the end-user based on the input/output of target KBS.

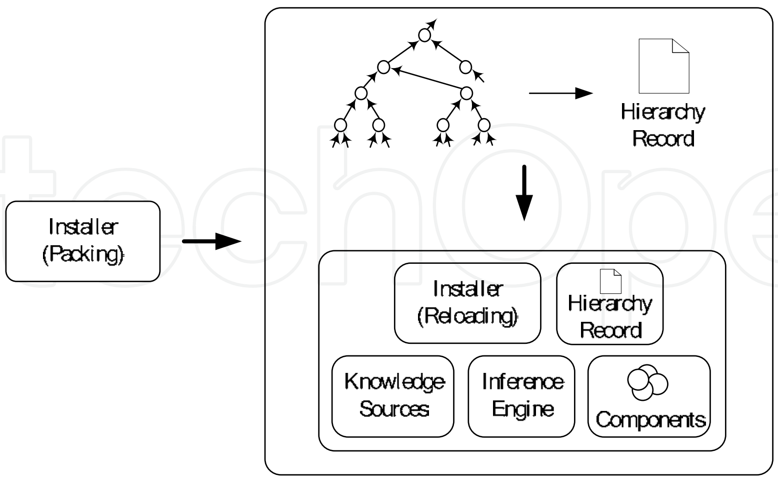


Fig. 4. Packing the KBS using Installer

Upon call to the target KBS received the installer will be started first to reload the KBS with all the necessary components, knowledge sources, and inference engine.

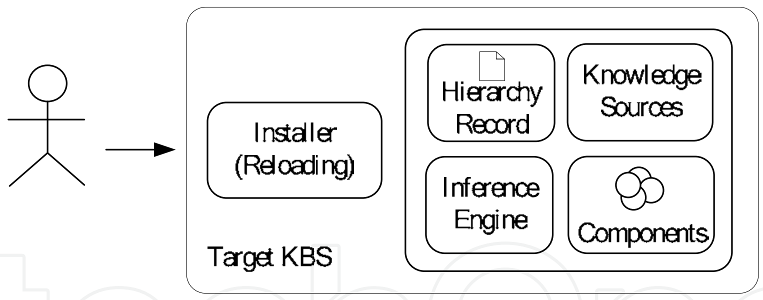


Fig. 5. Reloading the KBS using Installer

2.3. Components and Fields

The KWS warehouse stores pre-defined knowledge-based processing units that are the basic building blocks of KBS. Intelligent components are further classified by the nature of processing, in terms of the corresponding input and output. A KWS offers a set of k classes of intelligent components defined as

$$\boldsymbol{COM} = \{\boldsymbol{com}_1, \ldots, \boldsymbol{com}_k\},$$

and

$$\boldsymbol{com}_i = < cl_i\ t_i,\ s_i,\ c_i >,$$

where $i = 1, \ldots, k$, $cl_i \in \boldsymbol{CL} = \{cl_1, \ldots, cl_k\}$, the set of class names of intelligent components; $t_i \in \boldsymbol{T}_{cli}$, the *type* set under the class cl_i; $s_i \in \boldsymbol{S}_{cli}$, the *source and strategy* set under the class cl_i; and $c_i \in \boldsymbol{C}_{cli}$, the *connection* set under the class cl_i. At an *abstract* level, for any class *cl* defined, there is a mapping function f_{CL}:

$$f_{CL}: \boldsymbol{I}_{CL} \xrightarrow{K} \boldsymbol{O}_{CL}$$

where, $\boldsymbol{I}_{CL}$ is the input of the intelligent component of class *cl*, $\boldsymbol{O}_{CL}$ is the output, and K represents the corresponding knowledge-based processing. The features and properties of intelligent components under different classes are determined by their mapping function f_{CL}. It is an important feature of the KWS that an intelligent component under certain class always follows the same syntax for the interface with other intelligent components no matter which specific intelligent approach is adopted for the knowledge-based processing inside it. At the same time, intelligent components under the same class may behave differently when different approaches of knowledge-based processing are applied in problem solving. For instances, a decision-making may be done by applying traditional rule-based approach, or soft computing approaches, such as fuzzy logic inference, or neural networks; a knowledge discovery may be achieved by data mining applying different approaches; a prediction may be made by statistical methods or by using neural networks. When an intelligent component is defined as 'conditional component', it chooses suitable knowledge source to be applied among the alternatives provided according to run-time conditions detected. We have designed ten classes of intelligent components under two different categories: *processing components* and *learning components*, with each category including several classes based on the nature of function.

KWS also provides a possibility for the developers to include their own mathematical formulas or algorithms as user-defined procedures and make them intelligent components. Once such a procedure is defined, it becomes a special knowledge-based processing unit for possible use in other intelligent components in the same KBS under development.
Each of the input(s) and output of component is linked with a *Field*; fields are the basic data units indicated for input and output of processing intelligent components. They provide a pipeline for the data flow between components. An intelligent component can have multiple inputs, but only one output.
There are seven classes of processing component and three classes of learning component supported by KWS, as listed in Table 1.

Processing Component
Filter class applies its knowledge to check the input candidate list and filter out those "illegal" or "bad" members. $f_{Filtering} : I_{Filtering} \xrightarrow{K} O_{Filtering}$ 1. Where $\boldsymbol{I}_{\text{Filtering}}$ and $\boldsymbol{O}_{\text{Filtering}}$ are the input date set and output data set respectively, and $\boldsymbol{I}_{\text{Filtering}} \subseteq \boldsymbol{O}_{\text{Filtering}}$; 2. The input and output share the same type of data structure; 3. The length of output should not be longer than that of input;
Recognition class applies its knowledge to "read out" the meaning of a single input pattern. $f_{\text{Re}cognition} : p \xrightarrow{K} L$ 1. Where $\boldsymbol{p}$ is a single pattern, and $\boldsymbol{L} = \{l_1, \dots, l_k\}$ is a set of labels as possible recognition result, each of $l_i(1 \le i \le k)$ may be associated with a confidence value; 2. The input and output usually have different types of data structure; 3. The processing establishes one-to-one relation between an input pattern and an output label.
Summarization class contains the *Recognition* class as a special case, where the summary is a label or a highly summarized meaning. $f_{Summarization} : p \xrightarrow{K} P$ 1. Where $\boldsymbol{p}$ is a single input pattern, $\boldsymbol{P} = \{p'_1, \dots, p'_k\}$ is a set of patterns as possible summarization for the p, and each of p'_i $(1 \le i \le k)$ may be associated with a confidence value; 2. The input and output is equivalent or approximate in some degree, in terms of their meaning or explanation; 3. The degree or the level of abstraction of the output is determined by the knowledge applied and the inference mechanism adopted.
Confirmation checks the input, and gives "Yes/No" to each of the candidates. $f_{Confirmation} : D_c \xrightarrow{K} YN$ 1. Where $\boldsymbol{D}_c = \{d_1, \dots, d_k\}$ is a data set and $1 \le k$, $\boldsymbol{YN} = \{t_1, \dots, t_k\}$ is the corresponding truth list and i_i $(1 \le i \le k) \in \{Yes, No\}$; 2. It can be used as a conditional checker to support other intelligent components; 3. Fuzzy logic approaches may be introduced when a clear Yes/No cannot be simply decided.
Judgement is a more general class than *Confirmation* in the sense that the output can be defined as linguistic terms or values, such as high, expensive, going-up, or so. $f_{Judgement} : D_{JP} \xrightarrow{K} JP$ 1. Where $\boldsymbol{D}_{JP} = \{d_1, \dots, d_k\}$ is a data set and $1 \le k$; $\boldsymbol{JP} = \{term_1, \dots, term_k\}$ is a corresponding term list with possible confidence value associated, and $term_i(1 \le i \le k) \in \boldsymbol{LT}$, the set of pre-defined linguistic terms; 2. Conceptually, it contains the *Confirmation* class as a special case where the judgement is simply represented as Yes/No; 3. Changing the ***LT*** may change the behaviour of intelligent component.
Projection projects an input data set with k features to an output data set with $j \le k$ features.

$f_{Projection} : D_k \xrightarrow{K} D_j$ 1. Where $\boldsymbol{D}_k$ = {< $d^{(1)}_1, d^{(1)}_2, \ldots, d^{(1)}_k$ >,…, < $d^{(n)}_1, d^{(n)}_2, \ldots, d^{(n)}_k$ >} is an n-entry data set with k features, $\boldsymbol{D}_j$ = { < $d^{(1)'}_1, d^{(1)'}_2, \ldots, d^{(1)'}_j$ >,…, < $d^{(n)'}_1, d^{(n)'}_2, \ldots, d^{(n)'}_j$ >} is an n-entry data set with j ($j \le k$) features, and for any $1 \le i \le n$, the entry < $d^{(i)'}_1, d^{(i)'}_2, \ldots, d^{(i)'}_j$ > $\in \boldsymbol{D}_j$ is an image of the entry < $d^{(i)}_1, d^{(i)}_2, \ldots, d^{(i)}_k$ > $\in \boldsymbol{D}_k$ under the projection defined; 2. The process does not remove any data entry, but "remove" some of its features; 3. After projection, the data set will remain the entries but each of them appears in a space of probably lower dimension.
Decision checks the input as a situation and recommends a possible decision. $f_{Decision} : s \xrightarrow{K} AD$ 1. Where $\boldsymbol{s}$ is a single situation, and $\boldsymbol{AD}$ = { $ad_1, \ldots, ad_k$} is a list of recommended action or decision with possible confidence value associated; 2. This class of intelligent components is usually used at a late or final stage of intelligent systems, but not at the beginning; 3. For a complicated problem, multiple techniques and approaches may be required to form the inference strategy used in the component.
Learning Component
Discovery not only makes use of knowledge but also produces knowledge. It has relevant domain data for input and gives output as the knowledge discovered. $f_{Discovery} : D_D \xrightarrow{K} K_D$ 1. Where $\boldsymbol{D}_D$= { $d_1, \ldots, d_k$} is a data set and $1 \le k$; and $\boldsymbol{K}_D$ is a set of discovered knowledge of selected form, such as rules, relations, or other types; 2. Its output result can be applied as knowledge source to support other intelligent components; 3. It may use *Filtering* (a post-processing component's type) for its pre-processing or post-processing.
Training can train some rule on it based on the user input's data.
Post-Processing support *Learning Component* for post-processing.

Table 1. Intelligent component

2.4. KWS Inference Engine

The inference structure of a KBS constructed by KWS is represented as a knowledge hierarchy with multiple intelligent components connected. The task of construction can be done either by the intelligent editor or the KDL processor.

The knowledge hierarchy forms a static inference structure of the target KBS. A single intelligent component realizes the mapping from its input to its output with the support of its local knowledge base. The entire mapping of the KBS is achieved through the integration of intelligent components. There is no direct mapping relation from the input to the output of the KBS, but each intelligent component contributes to part of the mapping. Truth/confidence value is used to indicate uncertainty or imprecision that may occur in individual intelligent components, and also to connect the inference of individual components to the inference flow of the entire KBS.

One of the main challenges facing KWS for the construction of intelligent system is the complexity associated to inference mechanism having multi-level, and multi-modal knowledge integration. Each single intelligent component is actually a smaller KBS for a sub-problem of the target application, and its input and output can be directly linked to problem domain or the result from different stages of processing. How to assemble intelligent components to get a meaningful and unified data/information flow in the entire intelligent system constitutes a key task. Inference engine is necessary to control the execution which is realized through three aspects: 1) The management of protocol between components; 2) The control of execution order of components; and 3) The confidence transfer.

3. Truth Value Flow Inference

The concept of truth value flow inference (TVFI) was first put forward by Wang et al (P.Z. Wang and H.M. Zhang, 1993). It offers a conceptual mechanism of fuzzy inference in a network structure (L. Ding et al, 1996) and finds rationality in connection to the description of fuzzy propositions. Figure 6 shows a conceptual illustration of implication $P \rightarrow Q$ with TVFI, where C_P and C_Q are the confidence of *P* and *Q* respectively; *w* is the weight of rule (L. Ding and Z. Shen, 1994) controlling the channel of transferring the truth of *P* to the truth of *Q*.

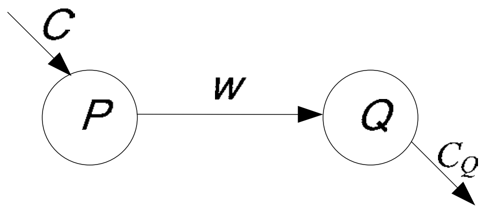


Fig. 6. Truth value flow inference

Based on the concepts of truth value flow inference and symbolical-numerical duality, we can construct a fuzzy inference with a static structure of nodes representing the relationship between propositions symbolically, and with a dynamic flow implementing the truth (or confidence) transfer among the individual nodes. This idea can be extended to KBS represented in network structure, with each intelligent component be treated as an extended node realizing a mapping relation between its input and output, and the entire KBS be an inference network (L. Ding and H.C. Lui, 1999).

3.1. Data Flow and Truth Value Flow in a Component

In order to have a unified inference flow in hybrid KBS, a possible solution is to separate the inference into a content level as well as a truth (confidence) level and handle them simultaneously. In this way, the content level of inference relies only on the knowledge sources stored "locally" in individual intelligent components whereas the truth (confidence) level of inference contributes to the flow of truth (confidence) throughout the entire system. So the inference in a hybrid KBS constructed by KWS is a truth value flow inference on the knowledge hierarchy.

In each intelligent component, two kinds of processing will be executed when receiving data in its inputs: content (or data) processing, and truth value (or confidence value) processing. These two types of processing are handled in intelligent component simultaneously to obtain the final result of the component that is the content of processing result associated with its corresponding confidence value. This concept can be shown as in Figure 7.

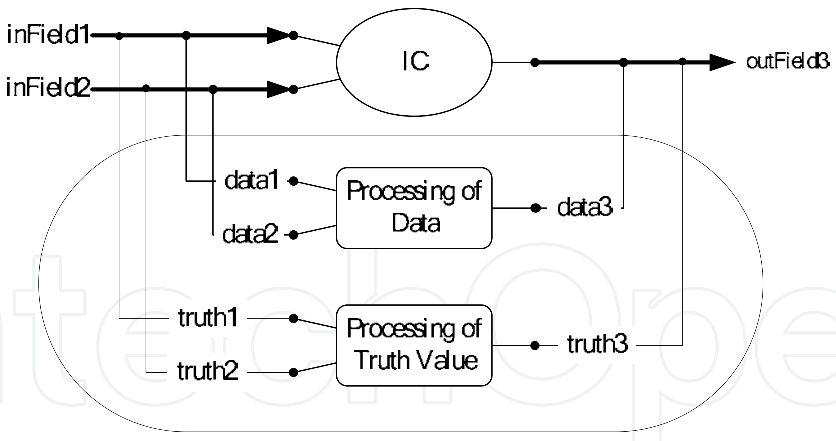


Fig. 7. An example of separated inputs in two levels

3.2. Interval-Valued Confidence

We adopted interval-valued confidence to represent the truth of fact and knowledge, and the confidence of inference. Here, the term "interval" is used in some different way from its usual meaning. Our motivation comes from several points.

1) A given truth value $t \in [0, 1]$ with some possible uncertainty or imprecision can always be understood as a fuzzy number defined on the closed interval [0, 1]. In an extreme case, t can be represented as a special fuzzy number with left, middle and right parameters at 0, t and 1 respectively, when its uncertainty reaches the maximum.
2) With more accurate information available, the range between the left and right parameters of such a fuzzy truth can be reduced. In another extreme case, we may have all the three parameters be t, if no uncertainty is considered, and it then comes back to usual case of single point of truth.
3) The acceptance of possible uncertainty associated with fuzzy truth is expected to allow more tolerance of imprecision in inference in terms of truth (confidence) calculation.

Three-parametric triangular truth value has some good features of easy representation and processing, intuitive interpretation consistent with common sense, convenient conversion from/to single-valued fuzzy truth, linguistic fuzzy truth, fuzzy numbers and fuzzy sets.

Definition 1 (general definition): A confidence value C of inference result is represented in the following general format:

$$C = (a, m, b), \tag{1}$$

where $0 \le a \le m \le b \le 1$, a is called the *left point*, b the *right point*, and m the *middle point*. It is a fuzzy subset defined on the universal set U which is the closed interval [0, 1], i.e., $C \subseteq U = [0, 1]$ (Figure 8-a).

Definition 2 (conversion of single-valued truth): A single-valued truth $t \in [0, 1]$ of a fact inputted from user is represented as: $T = (a, m, b)$ with the left point $a = t$, the right point $b = t$, and the middle point $m = t$ (Figure 8-b). It looks like a fuzzy singleton, but should be understood as a special case (t, t, t) of Definition 1.

Definition 3 (conversion of interval-valued truth): An interval-valued truth $[a, b]$, $0 \le a \le b \le 1$, of a fact inputted from user is represented as: $T = (a, m, b)$ with the middle point (Figure 8-c):

$$m = (a + b) / 2, \tag{2}$$

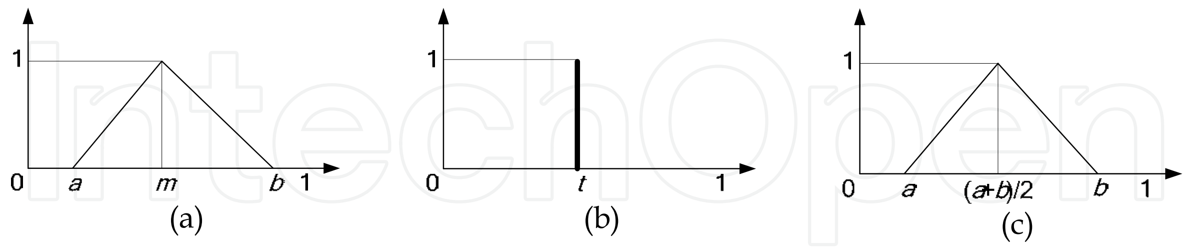


Fig. 8. Interval-Valued Confidence

Figure 9 shows the linguistic truth value true and false first given by L.A. Zadeh (L.A. Zadeh, 1975). Using the IVC defined in (1), we can represent them as: *true* = (0, 1, 1), and *false* = (0, 0, 1).

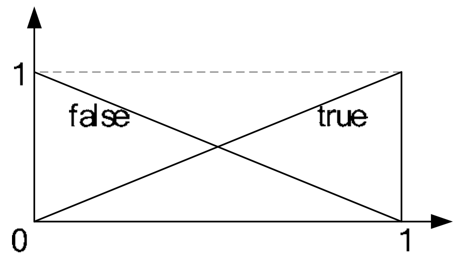


Fig. 9. Linguistic truth value *true* and *false*

Definition 4 (AND operation): The operation of AND on two interval-valued confidences $C_1 = (a_1, m_1, b_1)$ and $C_2 = (a_2, m_2, b_2)$, represented as in (1) is defined as

$$\begin{aligned}\mathbf{AND}_{\text{IVC}}(C_1, C_2) &= (C_a, C_m, C_b) \\ &= [\min(a_1, a_2), \min(m_1, m_2), \max(m_1, m_2)].\end{aligned} \tag{3}$$

Definition 5 (OR operation): The operation of OR on two interval-valued confidences $C_1 = (a_1, m_1, b_1)$ and $C_2 = (a_2, m_2, b_2)$, represented as in (1) is defined as

$$\begin{aligned}\mathbf{OR}_{\text{IVC}}(C_1, C_2) &= (C_a, C_m, C_b) \\ &= [\min(m_1, m_2), \max(m_1, m_2), \max(b_1, b_2)].\end{aligned} \tag{4}$$

Definition 6 (NOT operation): The operation of NOT on an interval-valued confidence $C = (a, m, b)$, represented as in (1) is defined as

$$\begin{aligned}\mathbf{NOT}_{\text{IVC}}(C) &= (C_a, C_m, C_b) \\ &= (1 - b, 1 - m, 1 - a).\end{aligned} \tag{5}$$

Applying the operations defined in Definitions 4, 5, and 6 on linguistic truth values *true* and *false*, we have:

$$\begin{aligned}
&\mathbf{OR}_{\mathrm{IVC}}(\mathit{true}, \mathit{false}) && (6)\\
&\quad = [\min(1, 0), \max(1, 0), \max(1, 1)] = \mathit{true},\\
&\mathbf{AND}_{\mathrm{IVC}}(\mathit{true}, \mathit{false}) && (7)\\
&\quad = [\min(0, 0), \min(1, 0), \max(1, 0)] = \mathit{false},\\
&\mathbf{NOT}_{\mathrm{IVC}}(\mathit{true}) && (8)\\
&\quad = (1-1, 1-1, 1-0) = (0, 0, 1) = \mathit{false},\\
&\mathbf{NOT}_{\mathrm{IVC}}(\mathit{false}) && (9)\\
&\quad = (1-1, 1-1, 1-0) = (0, 1, 1) = \mathit{true}.
\end{aligned}$$

The results of (6) ~ (9) are consistent with conventional definitions. However, we can also find that our operations provide interesting results when applying OR to both *true*, or AND to both *false*:

$$\begin{aligned}
&\mathbf{OR}_{\mathrm{IVC}}(\mathit{true}, \mathit{true}) && (10)\\
&= [\min(1, 1), \max(1, 1), \max(1, 1)] = (1, 1, 1),\\
&\mathbf{AND}_{\mathrm{IVC}}(\mathit{false}, \mathit{false}) && (11)\\
&= [\min(0, 0), \min(0, 0), \max(0, 0)] = (0, 0, 0).
\end{aligned}$$

We shall call the result (1, 1, 1) of (10) "*strong-true*", because it does not have any belief for "not-true", and the result (0, 0, 0) of (11) "*strong-false*" as it does not provide any possible room for "not-false". These two represent the extreme cases of IVC.
In order to have a further clear view of the properties of IVC with the corresponding operations, we list in Table 2 the typical results of $\mathbf{AND}_{\mathrm{IVC}}$, $\mathbf{OR}_{\mathrm{IVC}}$, and $\mathbf{NOT}_{\mathrm{IVC}}$ on *true*, *false*, *strong-true* and *strong-false*, represented in IVC format. The highlighted parts show that the results well meet commonsense interpretation.

Definition 7 (generalized AND): The operation of AND on k ($k > 2$) interval-valued confidences $C_1 = (a_1, m_1, b_1)$, …, $C_k = (a_k, m_k, b_k)$, represented as in (1) is defined as

$$\mathbf{AND}^{(g)}_{\mathrm{IVC}}(C_1, \ldots, C_k) = (C_a, C_m, C_b), \tag{12}$$

with C_a being the smallest value among the $a_1, \ldots, a_k$, C_m the smallest value among the $m_1, \ldots, m_k$, and C_b the second smallest value among the $m_1, \ldots, m_k$.

ANDIVC	F	T	s-F	s-T
F	F	F	s-F	F
T	F	T	F	T
s-F	s-F	F	s-F	F
s-T	F	T	F	s-T

(a) AND

$\mathrm{OR}_{\mathrm{IVC}}$	F	T	s-F	s-T
F	F	T	F	T
T	T	T	T	s-T
s-F	F	T	s-F	T
s-T	T	s-T	T	s-T

(b) OR

$\mathrm{NOT}_{\mathrm{IVC}}$	
F	T
T	F
s-F	s-T
s-T	s-F

(C) NOT

Table 2. Logical operations on *true*, *false*, *strong-true*, *string-false*

Definition 8 (generalized OR): The operation of OR on k ($k > 2$) interval-valued confidences $C_1 = (a_1, m_1, b_1)$, …, $C_k = (a_k, m_k, b_k)$, represented as in (1) is defined as

$$\mathbf{OR}^{(g)}_{\mathrm{IVC}}(C_1, \ldots, C_k) = (C_a, C_m, C_b), \tag{13}$$

with C_a being the second largest value among the $m_1, \ldots, m_k$, C_m the largest value among the $m_1, \ldots, m_k$, and C_b the largest value among the $b_1, \ldots, b_k$.
It is necessary to notice that in general the $\mathbf{OR}^{(g)}_{IVC}$ and $\mathbf{AND}^{(g)}_{IVC}$ operations do not satisfy connective laws as usual logic **OR** and **AND**. This can be seen from the following examples.

Example 1: Generalized $\mathbf{OR}^{(g)}_{IVC}$ and generalized $\mathbf{AND}^{(g)}_{IVC}$ operations do not satisfy connective laws.
Suppose $C_1 = (0.7, 0.9, 0.9)$, $C_2 = (0.6, 0.6, 1)$, and $C_3 = (0.4, 0.4, 1)$, with Definition 7 we have:

$$\mathbf{AND}^{(g)}_{IVC}\,(C_1, C_2, C_3) = (0.4, 0.4, 0.6). \tag{14}$$

However, we also have

$$\begin{gathered}\mathbf{AND}_{IVC}\,[\mathbf{AND}_{IVC}\,(C_1, C_2), C_3] \\ = \mathbf{AND}_{IVC}\,[(0.6, 0.6, 0.9), (0.4, 0.4, 1)] = (0.4, 0.4, 0.6),\end{gathered} \tag{15}$$

$$\begin{gathered}\mathbf{AND}_{IVC}\,[C_1, \mathbf{AND}_{IVC}\,(C_2, C_3)] \\ =\mathbf{AND}_{IVC}\,[(0.7, 0.9, 0.9), (0.4, 0.4, 0.6)]= (0.4, 0.4, 0.9).\end{gathered} \tag{16}$$

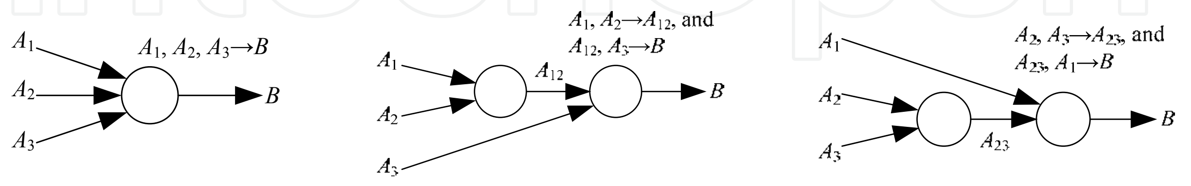
Similarly with Definition 8, we have

$$\mathbf{OR}^{(g)}_{IVC}\,(C_1, C_2, C_3) = (0.6, 0.9, 1). \tag{17}$$

$$\begin{gathered}\mathbf{OR}_{IVC}\,[\mathbf{OR}_{IVC}\,(C_1, C_2), C_3] \\ = \mathbf{OR}_{IVC}\,[(0.6, 0.9, 1), (0.4, 0.4, 1)] = (0.4, 0.9, 1),\end{gathered} \tag{18}$$

$$\begin{gathered}\mathbf{OR}_{IVC}\,[C_1, \mathbf{OR}_{IVC}\,(C_2, C_3)] \\ = \mathbf{OR}_{IVC}\,[(0.7, 0.9, 0.9), (0.4, 0.6, 1)] = (0.6, 0.9, 1).\end{gathered} \tag{19}$$

The highlighted parts show the difference between (15) and (16), and between (18) and (19). This feature can be interpreted by the truth value flow inference adopted in KWS. It is understood that the corresponding structures of TVFI for (14), (15) and (16) are different (Figure 10), and the same applies to (17), (18) and (19). Figure 10 gives three structures of TVFI. The nodes of the structures are representing intelligent components for knowledge-based processing, and handling truth value flow from the input side to the output side. So the (a), (b), and (c) actually represent different *internal* inference flows, though they have the same input interface A_1, A_2 and A_3, and output interface B for the entire structure.



A_1
$A_1, A_2, A_3 \rightarrow B$
A_2
B
A_3
$A_1, A_2 \rightarrow A_{12}$, and
$A_{12}, A_3 \rightarrow B$
A_1
A_{12}
B
A_2
A_3
A_1
$A_2, A_3 \rightarrow A_{23}$, and
$A_{23}, A_1 \rightarrow B$
A_2
A_3
A_{23}
B

(a) $\mathbf{AND}^{(g)}_{IVC}\,(A_1, A_2, A_3)$ (b) $\mathbf{AND}_{IVC}\,[\mathbf{AND}_{IVC}\,(A_1, A_2), A_3]$ (c) $\mathbf{AND}_{IVC}\,[A_1, \mathbf{AND}_{IVC}\,(A_2, A_3)]$
Fig. 10. TVFI structures

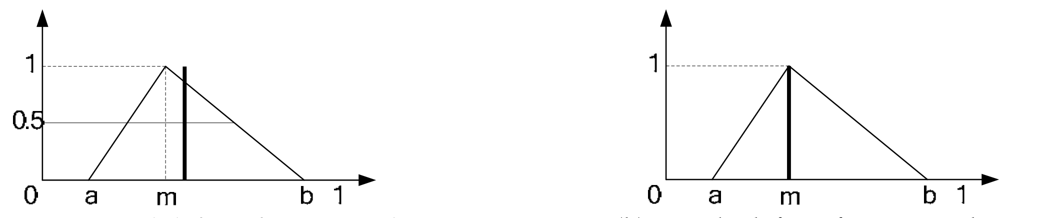
When a single-valued confidence of conclusion is desired, we need to consider defuzzification in the last stage of inference in hybrid KBS. As defuzzification is considered a matter of application-specific, we here propose two simplified calculations based on the idea of conventional center of gravity approach (J.-S.R. Jang et al, 1997; R.R. Yager and D.P. Filev, 1994) with the reference to the left, middle and right points of IVC.

Definition 9 (compromised defuzzification): The compromised defuzzification $\mathbf{Def}_{\mathrm{com}}(C)$ of IVC $C = (a, m, b)$, $0 \le a \le m \le b \le 1$, is defined as the center of gravity of α-cut with α= 0.5 (Figure 11-a).
Since the IVC is a fuzzy number defined as a piece-wise linear function with the corresponding left, middle, and right points, we have the following calculation:

$$\mathbf{Def}_{\mathrm{com}}(C) = [(a + m)/2 + (m + b)/2] \,/\, 2. \tag{20}$$

Definition 10 (simple defuzzification): The simple defuzzification $\mathbf{Def}_{\mathrm{sim}}(C)$ of IVC $C = (a, m, b)$, $0 \le a \le m \le b \le 1$, is defined as the middle point m (Figure 11-b):

$$\mathbf{Def}_{\mathrm{sim}}\,(C) = m. \tag{21}$$



(a) Compromised defuzzification with IVC (b) Simple defuzzification with IVC
Fig. 11. Defuzzification with IVC

Example 2: Single-valued confidence and interval-valued confidence with corresponding AND/OR operators.

A. Single-Valued Confidence using Min/Max for AND/OR operation
Consider the following rule and facts given:

rule 1:	if the topic is interesting,
	and the weather is good,
	then I will attend the seminar;
fact 1:	the topic is interesting;
fact 2:	the weather is good.

The most common way of handling ***and*** is to use *min* as *t*-norm to calculate the overall truth of the premise from the two subpremises, and when both facts are 0.5 true, for instance, we get *min*(0.5, 0.5) = 0.5. Now let's consider the fact about interesting topic:

fact 1′: the topic is interesting (0.9 true).

With *min*, we will still get the same truth 0.5 for the premise, i.e., the influence of interesting topic has been buried by the fact of weather as long as its truth is not lower than the other fact. However, we tend to agree that a more interesting topic (0.9) makes a person more willing to go to the seminar than a moderately interesting topic (0.5) given the same weather condition (0.5). The following example shows a similar problem with ***or*** operation.

rule 2: if Mr. A and Mr. B are first cousin,
or second cousin,
then they have a close relationship;
fact 3: Mr. A and Mr. B are first cousin;
fact 4: Mr. A and Mr. B are second cousin.

With *max*, the most common way of calculating *t*-conorm, when either fact 3 or fact 4 is 0.5 true and the other is 0 true, we can obtain the overall truth of premise to be *max*(0.5, 0) = 0.5. However, it is also possible that Mr. A and Mr. B have both first cousin and second cousin relationship when one's parents being first cousin. Assume both subpremises with 0.5 confidence, we will still have the same 0.5 for the truth of premise using max calculation. Conventionally, two persons that are in both first cousin and second cousin relation should more likely to have a close relationship than only being one kind of cousin having their double connections of relative. Obviously, *max* does not well reflect this situation.
From the above examples, we can see that a single-valued truth (confidence) does not provide sufficient room for the description of imprecise knowledge, especially in decision making applications, where subjective knowledge and experience play an important role and the truth of subjective knowledge is hardly to be measurable in absolute sense. It is also very often that in real applications, a single-valued truth (confidence) does not necessarily mean in the explicit way as it seems. For instance, when a user inputs 0.8 as the truth of good weather, it should not be simply treated as a precise value 0.8 but some thing *around* 0.8.

B. Interval-Valued Confidence using AND_{IVC}/OR_{IVC}
We apply the IVC and corresponding operations to the previous examples. For fact 1 and fact 2, we have

$\mathbf{AND}_{\mathrm{IVC}}$ [(0.5, 0.5, 0.5), (0.5, 0.5, 0.5)]
= [min(0.5, 0.5), min(0.5, 0.5), max(0.5, 0.5)]
= (0.5, 0.5, 0.5),

and for fact 1′ and fact 2, we have

$\mathbf{AND}_{\mathrm{IVC}}$[(0.9, 0.9, 0.9), (0.5, 0.5, 0.5)] (13)
= [min(0.9, 0.5), min(0.9, 0.5), max(0.9, 0.5)]
= (0.5, 0.5, 0.9).
It shows that fact 1′ together with fact 2 gives more potential to have a truth higher than 0.5 (Figure 12).
For fact 3 (0.5 true) or fact 4 (0 true), we have

$\mathbf{OR}_{\mathrm{IVC}}$[(0.5, 0.5, 0.5), (0, 0, 0)]
= [min(0.5, 0), max(0.5, 0), max(0.5, 0)]
= (0, 0.5, 0.5),

and for fact 3 (0.5 true) or fact 4′ (0.5 true), we have

$\mathbf{OR}_{\mathrm{IVC}}$[(0.5, 0.5, 0.5), (0.5, 0.5, 0.5)]
= [min(0.5, 0.5), max(0.5, 0.5), max(0.5, 0.5)]
= (0.5, 0.5, 0.5).

It shows that fact 3 together with fact 4′ has a stronger belief for truth 0.5 (Figure 13).

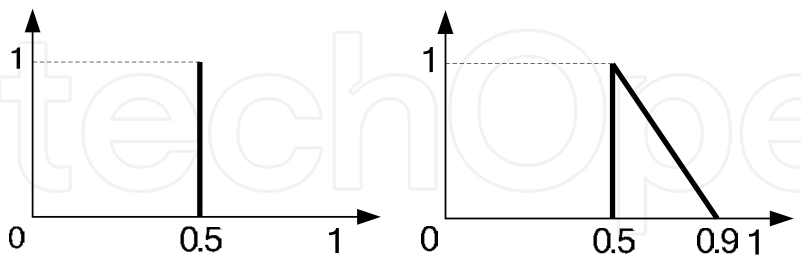


Fig. 12. An example of $\mathbf{AND}_{\mathrm{IVC}}$

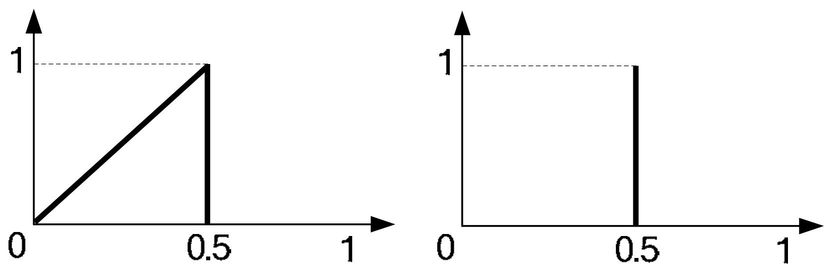


Fig. 13. An example of $\mathbf{OR}_{\mathrm{IVC}}$

From above discussion, we can see that using IVC and the corresponding operations defined for confidence calculation, partial conclusion with a relatively stronger confidence about true or false will not easily make the influence of other parts be totally ignored in inference.

3.3. Confidence Transfer and Interpretability

The processing in intelligent component can be further classified in several categories according to the description of mapping relation.

3.3.1. Component with Interpretable Mapping Relation

When mapping relation between input and output of intelligent components can be interpreted by a mathematic formula or an algorithm described by procedure. The mapping relation is considered interpretable, and precise in the sense that the processing does not affect uncertainty and imprecision. In this type of intelligent components, the confidence of output remains the same as that of input.

Example 3: A component performs some simple data processing, such as sorting. The output of component is the sorted result based on certain condition specified with knowledge source. In this case, the mapping relation between input and output can be determined by algorithm, and the corresponding output truth value remains the same as the input truth value.

If a component uses rule-based knowledge (fuzzy or precise) that can be approximately interpreted by **AND**, **OR**, and **NOT** relations, the mapping relation is also a kind of interpretable, but the truth of output may be affected by the knowledge-based processing. The corresponding truth value of output can be determined by the logic relations used in the rules.

3.3.2. Component with Less Interpretable Mapping Relation

When an IC uses less interpretable knowledge representation, e.g., neural networks, or case-based reasoning, the mapping relationship realized by the IC may not be interpreted in composition of logic operations and therefore the input confidence of the IC cannot be simply transferred to its output side to obtain the output confidence through its internal inference structure (L. Ding and S.L. Lo, 2008). A further extension of the framework of truth value flow inference using IVC (L. Ding, 2008) is needed to cope with this problem. It is achieved by two steps:

1) First carry out the internal inference of such an IC by assuming that the input is completely true (i.e. with full confidence);
2) Combine the input confidence with the result confidence as one unified output confidence at the output side of an IC after its processing.

We adopt the concept of truth base introduced with the exponential form of fuzzy logic (Z. Shen and L. Ding, 1994) for the interpretation of confidence transfer.

A. Truth base and confidence representation

The exponential form of fuzzy logic (EF) was proposed for confidence comparison and high order fuzziness simplification (Z. Shen and L. Ding, 1994). It provides a possible way for confidence transfer in intelligent components that use less interpretable representation of knowledge. An important concept introduced with EF is the truth base. For instance, saying "*P* is 0.8 true" may be understood in two ways: "*P* has *complete truth* (1) with 0.8 confidence", or "*P* has 0.8 truth with *full confidence* (1)". The difference is from the use of different truth bases: in the former, we put our truth base at 1, whereas in the latter, we put our truth base at 0.8. Obviously, it is reasonable to make these two ways of understanding be exchangeable from one to the other equivalently.

Usually by default we take *completely true* as the basis of our discussion about confidence, e.g.: 1 in fuzzy valued logic, or true in fuzzy linguistic valued logic, but it is also useful to have a different truth base for the convenience of discussion and have confidences of different truth bases be convertible from one to other.

The EF is originally defined with both *truth*-**I** and *truth*-**II** of fuzzy valued logic and fuzzy linguistic valued logic (Z. Shen and L. Ding, 1994). In KWS *truth*-**I** of fuzzy valued logic is adopted.

Definition 11 (EF on fuzzy valued logic): Let $t \in [0, 1]$ be a truth value in fuzzy valued logic, then t can be represented in its exponential form B^c when

$$t = (B - \mathbf{U}) \times c + \mathbf{U}, \tag{22}$$

where B ∈ [0, 1] is called the *fuzzy truth base*, $c \in (-\infty, \infty)$ is called *confidence exponent*, **U** is the unknown point for inference. In *truth*-**I**, we further specify **U** = 0, and B∈(0, 1].
It is important to be aware of that a super confidence $c > 1$ may cause a loss of information in inference (Z. Shen and L. Ding, 1994), so a truth base $B \geq t$ is usually recommended.
When applying the EF originally defined with single truth value to IVC, we have the *IVC format of unknown* $\mathbf{U}_{\text{IVC}} = (0, 0, 0)$, the *IVC format of truth base* $\boldsymbol{B}_{\text{IVC}} = (B, B, B)$ with $B \in (0, 1]$, and the IVC format of confidence exponent $C = (a_c, m_c, b_c)$. So the above (22) can be rewritten as:

$$t_{\text{IVC}} = (a_t, m_t, b_t) = (B \times a_c, B \times m_c, B \times b_c). \tag{23}$$

Definition 12 (Base changing in EF): The exponential form of a fuzzy truth t on truth base B_1 can be converted to that on truth base B_2 by

$$t = B_1^{C_1} = B_2^{C_2}. \tag{24}$$

where $B_1, B_2 \neq 0$, **U** is the unknown point of inference, $B_1, B_2 \neq \mathbf{U}$, and c_1, c_2, B_1 and B_2 satisfy the following relation:

$$c_2 = c_1 \times (B_1 - \mathbf{U}) \div (B_2 - \mathbf{U}). \tag{25}$$

Using the IVC format of truth base and unknown point, given two confidences $C_1 = (a_1, m_1, b_1)$ under truth base $B_1 = (B_1, B_1, B_1)$ and $C_2 = (a_2, m_2, b_2)$ under $B_2 = (B_2, B_2, B_2)$, the above (23) can be rewritten as:

$$C_2 = (a_2, m_2, b_2) = (a_1 \times B_1 \div B_2, m_1 \times B_1 \div B_2, b_1 \times B_1 \div B_2). \tag{26}$$

Definition 13 (Logical operations on EF): The AND, OR and NOT operations on EF are defined as:

$$\textbf{AND}(B^{C_1}, B^{C_2}, \ldots, B^{Cn}) = B^{\text{AND}(c_1, c_2, \ldots, c_n)} \tag{27}$$
$$\textbf{OR}(B^{C_1}, B^{C_2}, \ldots, B^{Cn}) = B^{\text{OR}(c_1, c_2, \ldots, c_n)} \tag{28}$$
$$\textbf{NOT}(B^{C}) = B^{\text{NOT}(c)} \tag{29}$$

where B is a given *common truth base*, and EF values originally with different truth bases are converted to the selected *common truth base* before carrying out logical operations.

B. Confidence transfer with arbitrary intelligent component
Assume an arbitrary intelligent component A with $m \geq 1$ input variables $\text{in}_1, \text{in}_2, \ldots, \text{in}_m$ (Figure 14) without loss of generality, where K represents a knowledge-based mapping realized in this component. The input in_k ($1 \leq k \leq m$) is denoted by $<d_k, B_k^{C_k}>$, where the data d_k is from other intelligent component IC-k, and associated with IVC $C_k = (a_k, m_k, b_k)$ under a selected truth base B_k. When a common truth base $\boldsymbol{B}$ is selected for all the inputs $\text{in}_1, \text{in}_2, \ldots,$

in$_m$, can be represented in its simplified form by $<d_k, C_k>$ without confusion caused. It will be considered as a special case of having an empty IC when d_k is directly from problem domain. The inference output of *A* is obtained through the following algorithm.
In a hybrid KBS constructed by KWS, the *default common truth base* is set as ***T*** = (1, 1, 1), the *strong true* in IVC format. This also applies to intelligent components with interpretable rule-based type of knowledge, and so the discussion of confidence transfer in (L. Ding, 2008) can be rebuilt using EF with the default common truth base.

Algorithm-1 (Confidence transfer of IC uses less interpretable knowledge representation):

1) The input $<d_k, B_k^{C_k}>$ ($1 \le k \le m$) is first converted to $<d_k, T^{C_{in-k}}>$, where ***T*** is the *strong true* (1, 1, 1) in IVC format, and $C_{in-k} = (a_{in-k}, m_{in-k}, b_{in-k})$ is the confidence exponent in IVC format, through base changing;
2) The *combined confidence of input* is then calculated by
   $C_{in} = (a_{in}, m_{in}, b_{in}) = [\min_k(a_{in-k}), \min_k(m_{in-k}), \min_k(b_{in-k})]$;
3) The data $d_1, \ldots, d_k, \ldots, d_m$ are then accepted as input values with perfect confidence for the inference in *A*;
4) Assume that a data *r* is obtained as the content of inference result of *A* with the result confidence B_r^{Cr}, where $C_r = (a_r, m_r, b_r)$. The $T^{Cr*}$is converted to B_r^{Cr} through base changing, then the output confidence of *A* is calculated by
   $C_{out} = \mathbf{AND}_{IVC} (C_{in}, C_r^*)$
   based on the Definition 4, and $<r, T^{C_{out}}>$ is the output of *A*

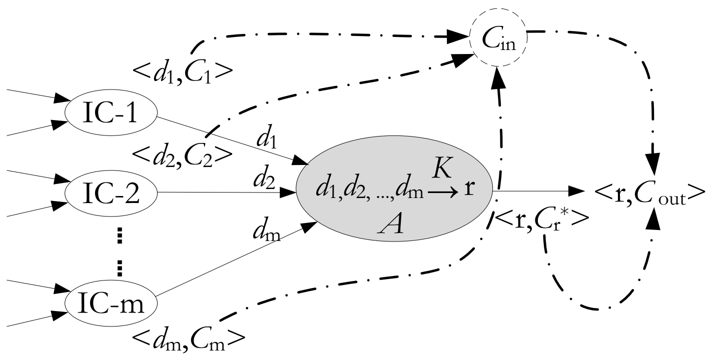


Fig. 14. Confidence transfer in an IC of hybrid KBS

3.3.3. Confidence Transfer in Hybrid KBS

Compared with a typical fuzzy inference system (J.-S.R. Jang et al, 1997; R.R. Yager, 1994), a hybrid KBS constructed by KWS usually does not have a universal knowledge base but multiple knowledge sources associated in individual intelligent components. In this sense, each knowledge source has only a local affection to the corresponding intelligent component realizing a mapping relation between its input and output. Given two arbitrary intelligent components A and B, having the output of A linked to the input of B means its content is passed on for further knowledge-based processing in B, and at the same time its confidence is integrated in the calculation of the confidence of output of B. Therefore, an IC is needed to distinguish the uncertainty associated with external input or introduced by its internal inference result. We represent the former as input confidence, and the latter as result confidence. When all the intelligent components in a hybrid KBS use rule-based knowledge (fuzzy or precise) that can be approximately interpreted using AND, OR, and NOT relations, the inference of the KBS can be interpreted as a confidence flow on an extended logic-based network that embeds internal inference structure of individual IC into the knowledge hierarchy (L. Ding et al, 1996; L. Ding, 2008).

Example 4: With the rapid development of Internet technologies, people are receiving more and more e-mails for commercial promotion purpose and often need spend time and effort for filtering and cleaning. We consider a simple hierarchical KBS for the filtering function based on the title of e-mail. There are three major parts of title related to promotion: action (of promotion), e.g., "offer", "provide", or "sale"; benefit (to user), e.g., "saving", "earn", or "win"; currency (or percentage, number), e.g., "$", "%", or "xx.xx".
The system consists of five intelligent components of type summarization, recognition, and decision (L. Ding, 2007b; L. Ding and S. Nadkarni, 2007) (Fig. 15).

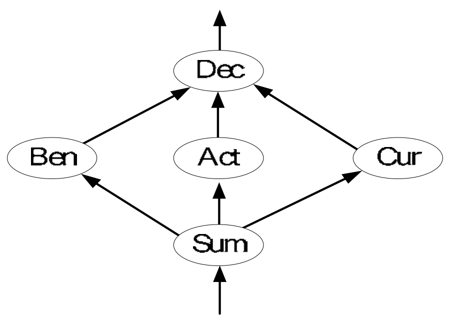


Fig. 15. An example of hierarchical inference with IVC flow

Sum (S) - a summarization component that performs pre-processing to eliminate less relevant words, such as "a","and", and so on, indicated in a knowledge source of dictionary. The cleaned-up version of email title will be passed up to three recognition components for further processing.
Act (***A***) - a recognition component to recognize words that match the "action" category. The knowledge source defines the kind of words often used to positively describe promotion action, including the representative words and their major variants.
Ben (***B***) - a recognition component to recognize words that match the "benefit" category. The knowledge source defines the kind of words often used to highlight the potential benefit to attract people's attention, including the representative words as well as their major variants.
Cur (***C***) - a recognition component to recognize characters of currency, percentage, or numbers.
Dec (***D***) - a decision component to decide whether the text examined is suspicious for a promotion with the combined results from ***A***, ***B***, and ***C***. The decision knowledge may be fuzzy association rules obtained through possible knowledge discovery, such as:

rule-d1: If A **and** B, Then P (0.8)
rule-d2: If B **or** C, Then P (0.4)
rule-d3: If A **and** C, Then P (0.6)

When an e-mail is received with a title like:

"Special Offer for … Saving up to 99% …", the corresponding processing steps will be:

(a) ***S*** filters out the less relevant words and obtained a cleaned-up version:

"Offer Saving 99%"

(b) ***A*** recognizes "Offer" as an action word with a match in IVC (0, 1, 1).

(c) ***B*** recognizes "Saving" as a variant of benefit word"save" with IVC (0, 0.8, 1).

(d) ***C*** recognizes "%" as a percentage character with confidence (0, 1, 1) as well as number "99" with confidence (0, 1, 1). Assume the knowledge of ***C*** is defined as:

If \$or%-character **or** number then is-C.

So we have overall confidence for ***C*** is

$\mathbf{OR}_{\mathrm{IVC}}[(0, 1, 1), (0, 1, 1)] = (1, 1, 1)$.

(e) ***D*** combines the results from ***A***, ***B***, and ***C***. Here, the confidence of conclusion is defined as **and** (confidence-of-premise, confidence-of-rule). We check each of the rules.

d1: confidence-of-premise
$= \mathbf{AND}_{\mathrm{IVC}}[(0, 1, 1), (0, 0.8, 1)] = (0, 0.8, 1)$;
confidence-of-conclusion
$= \mathbf{AND}_{\mathrm{IVC}}[(0, 0.8, 1), (0.8, 0.8, 0.8)] = (0, 0.8, 0.8)$.

d2: confidence-of-premise
$= \mathbf{OR}_{\mathrm{IVC}}[(0, 0.8, 1), (1, 1, 1)] = (0.8, 1, 1)$;
confidence-of-conclusion
$= \mathbf{AND}_{\mathrm{IVC}}[(0.8, 1, 1), (0.4, 0.4, 0.4)] = (0.4, 0.4, 1)$.

d3: confidence-of-premise
$= \mathbf{AND}_{\mathrm{IVC}}[(0, 1, 1), (1, 1, 1))] = (0, 1, 1)$;
confidence-of-conclusion
$= \mathbf{AND}_{\mathrm{IVC}}[(0, 1, 1), (0.6, 0.6, 0.6)] = (0, 0.6, 1)$.

(f) Now we aggregate the results of d1~d3 from (e):

$\mathbf{OR}^{(g)}{}_{\mathrm{IVC}}[(0, 0.8, 0.8), (0.4, 0.4, 1), (0, 0.6, 1)]$
$= (0.6, 0.8, 1)$.

Therefore, this e-mail is a commercial promotion with a high possibility 0.8 (if defuzzification is applied) or "*very likely*" as a linguistic interpretation.

Example 5: We replace the component ***Dec (D)*** in Example 4 with the below:
Dec2 (***D2***) - a decision component to decide whether the text under check is suspicious for a promotion with the combined results from ***A***, ***B***, and ***C***. The decision knowledge used is case-based reasoning technique.
When an e-mail is received with a title like:
"Special Offer for … Saving up to 99% …", the corresponding processing steps will be:

(a) ~ (d), the same as in Example 4;

(e) ***D2*** combines the results from ***A***, ***B***, and ***C***. Using Algorithm-1 given in Section 4.3.2, we have:

Step-1: Result from ***A*** <"Offer", (0, 1, 1)> converted to <"Offer", $(1, 1, 1)^{(0, 1, 1)}$>;
Result from ***B*** <"Saving", (0, 0.8, 1)> converted to <"Saving", $(1, 1, 1)^{(0, 0.8, 1)}$>;
Result from ***C*** <"99%", (0, 1, 1)> converted to <"99%", $(1, 1, 1)^{(0, 1, 1)}$>.

Step-2: The combined confidence of input is then calculated by

$C_{in} = (a_{in}, m_{in}, b_{in})$
$= (\min(0, 0, 0), \min(1, 0.8, 1), \min(1, 1, 1))$
$= (0, 0.8, 1)$

Step-3: The data <"Offer", "Saving", "99%"> are then accepted as input values with perfect confidence for the inference in ***D2***

Step-4: Assume that component ***D2*** through case-based reasoning technique to get the result <"Promotion", (0, 0.9, 1)>, we have:
$C_r = (0, 0.9, 1)$ and $C_{in} = (0, 0.8, 1)$
Then the output confidence of ***D2*** is calculated by
$C_{out} = \mathbf{AND}_{IVC}\,(C_{in}, C_r^*)$
$= (0, 0.8, 0.9)$

(f) Finally, we have the result <"Promotion", $(1, 1, 1)^{(0,\ 0.8,\ 0.9)}$> and converted to <"Promotion", (0, 0.8, 0.9)>, it is the final result.

Therefore, this e-mail is a commercial promotion with a high possibility 0.8 (if defuzzification is applied).

4. KWS Inference Engine

As mentioned previously, KDL processor is to construct the knowledge hierarchy that forms the static inference structure of target KBS. The execution on such a static inference structure of KBS is carried out layer by layer in bottom-up manner. An inference engine is needed to control the execution of components in KBS by managing protocol between components, and sending necessary signals for the order of execution. A component in an inference structure constructed by KWS is a customized knowledge-based processing unit, and a field in the inference structure is a space that stores input data or intermediate result during inference. Fig. 16 gives an example of inference structure.

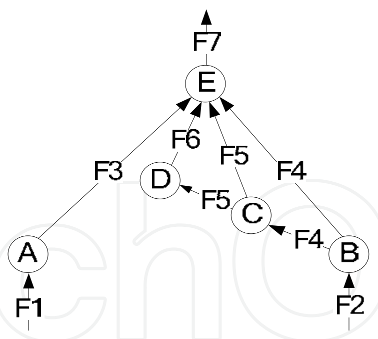


Fig. 16. An example of inference structure

4.1. Level of Component and Layer of Field

Based on the position of each component in inference hierarchy, a topological sorting determines the execution order with which a child component should always be executed before its parent component. For the purpose of such topological sorting, we need to first introduce two concepts: level of component, and layer of field. Algorithm-2 is to determine the level of all the components in a given knowledge hierarchy, as well as the layer of each of the fields associated with the components.

Algorithm-2 (Determine level of component and layer of field):
1) For a field **f** that receives input data directly from application, set layer(**f**) = 1, where layer(**f**) is the layer of **f**;
2) For a field **f** that serves as the output field of component **C**, set layer(**f**) = level(**C**) + 1, where level(**C**) is the level of **C**;
3) For a component **C** with h ($h \geq 1$) input fields $\mathbf{f}_{c1}$, $\mathbf{f}_{c2}$, ..., $\mathbf{f}_{ch}$, set level(**C**) = max[layer($\mathbf{f}_{c1}$), layer ($\mathbf{f}_{c2}$), ..., layer ($\mathbf{f}_{ch}$)].

The level of components and the layer of fields in the example given in Figure 16 are shown in Table 3-a and Table 3-b, respectively.
Except usual tree structure, in inference structure there are some special graph structures which need special handling by inference engine: 1) multiple parents, and 2) cross layer. For example, in Figure 16 component ***D*** and ***E*** are the parents of component ***C***, component ***B*** of level-1 passes its result to component ***E*** of level-4. The order of execution is determined by a topological sorting according to level of components. The key issue here is the data consistency.

Component	Level
A, B	1
C	2
D	3
E	4

(a) The level of components

Field	Layer
F1, F2	1
F3, F4	2
F5	3
F6	4
F7	5

(b) The layer of fields

Table 3. The level of components and the layer of fields for Figure 16

4.2. Protocol between Components

The protocol between components is described from three aspects: syntax, semantics and data type. With the general classes of intelligent components defined, we have syntactical rules indicating the possible connections between different classes. For instance, a component of *Confirmation* class is allowed to send its output to the input of a component of *Decision* class, but not allowed to do the same to the input of a component of *Filtering* class.
For each allowable connection between classes, we further set semantic rules with more details to specify legal connections. A *Filtering* component may connect to another Filtering component syntactically. However, there may be semantic constraints based on the detailed types of knowledge used in each *Filtering* component. For instance, a *Dictionary* component can be the support (child component) for a *List* component, but the reverse case does not hold true.
At the component-to-component level, there are four kinds of protocol for the data type of implementation.
1) single-to-single: a singleton data is connected to an input field of singleton.
2) single-to-multiple: a singleton data is connected to an input field of vector.
3) multiple-to-single: a vector data is connected to an input field of singleton.
4) multiple-to-multiple: a vector data is connected to an input field of vector.

4.3. Forward Inference with Partial Feedback

It is always desired to get a "better" solution when knowledge-based processing involved in an intelligent component can provide multiple candidates of solution for output. In order to fulfil this purpose, the inference in KBS constructed by KWS is a forward inference with partial feedback. When a component receives inputs, it executes and generates the result as output. As a typical scenario, a component generates inference result and passes the result to its parent(s), and receives *Rerun* signal from parent(s) to provide next possible result when the previously submitted result is found unsatisfactory. The *rerun* mechanism provides a possible way to extend the forward inference mechanism in KBS. Final result will only be generated when the inference is successful.

The inference in KBS constructed by KWS is basically a forward inference. As the simple case when there is no feedback considered, the inference flow starts from layer-1 receiving input data directly from application, goes up for the level-1 components to execute and provide result as layer-2, and then further goes up for the level-2 components to execute, …, finally has the last level components execute to provide result as the last layer, which represents the inference result.

For a more general case when there is partial feedback introduced, if a component of level-*k* (*k* = 2, 3, …) finds some of the input from its child component unsatisfactory, it will send a *Rerun* signal to the corresponding child component, and the current execution will be pulled back down to the level of the child component accordingly. When there are several components send *Rerun* signal to their child components, the current execution will be set as the level that is the lowest among the levels of components that received *Rerun* signal.

Considering again the example given in Fig. 16, if component ***C*** sent a *Rerun* signal to component ***B***, then the current execution will be pulled back to level-1 for ***B*** to execute its function again to generate next possible results. With a similar spirit, if component ***E*** sent a *Rerun* signal to component ***B***, then the current execution will also be pulled back to level-1. It is important to notice that there are other two components ***C*** and ***D*** at a higher level than ***B***, and with the *Rerun* signal sent to ***B***, any execution starting from ***C*** and ***D*** will be frozen temporarily to ensure the data consistency.

4.3.1 States of Component

In order to indicate the execution status of a component, we introduce *state of component*. The transition between states is shown in Fig. 17 and the explanation is listed in Table 4.

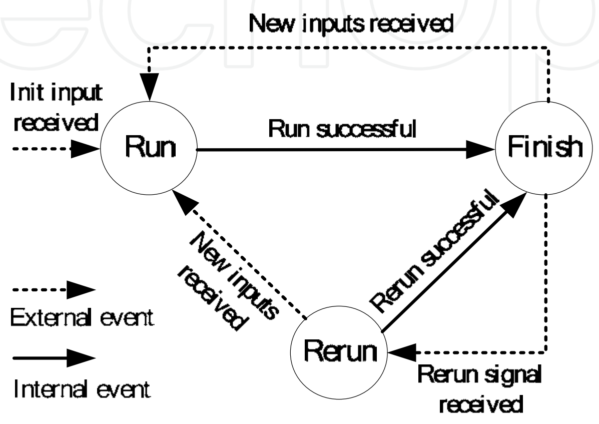


Fig. 17. The states of a component

State	Explanation
Run	The component will execute its function, if successful; the results will be sent to its parent, if unsuccessful, the component will send a rerun signal to its child components.
Rerun	The component will execute its rerun function, trying to generate next possible results. If successful, the results will be sent to its parent, if unsuccessful, the component will send a rerun signal to its child components.
Finish	The run or rerun of component is successful.

Table 4. The explanation of component states

The state of an output field as same as the corresponding component which sends result to the field, the explanation of field states is listed in Table 5.

State	Explanation
Run	Representing a field "waiting for obtaining result".
Rerun	Representing an output field "waiting for obtaining new result of rerun".
Finish	Representing a field "finished obtaining result".

Table 5. The explanation of field states

Example 6: Consider a scenario of execution on the inference structure given in Fig. .

Time-1: The current execution is at level-1, and Field F1 and F2 received inputs from layer-1, and component ***A*** and ***B*** executed and provided result as layer-2, finally, component ***A*** and ***B*** updated their state to be "Finish", and the current execution is updated to be at level-2.

Time-2: The current execution is at level-2. Assume that the result from component ***B*** is unsatisfactory for component ***C***. ***C*** sent the *Rerun* signal to ***B*** to get next possible inputs, ***B*** received the *Rerun* signal, and its state is updated to "Rerun", and the current execution is pulled back down to level-1.

Time	Level	Component				
		A	B	C	D	E
1	1	R	R	R	R	R
2	2	F	F	R	R	R
2	Assume that, C sent rerun signal to B					
3	1	F	RR	R	R	R
3	Assume that, B generate next possible result					
4	2	F	F	R	R	R
5	3	F	F	F	R	R
6	4	F	F	F	F	R
6	Assume that, E sent rerun signal to B, C, D					
7	1	F	RR	RR	RR	R
7	Assume that, B generate next possible result					
8	2	F	F	R	RR	R
9	3	F	F	F	R	R
10	4	F	F	F	F	R
11	5	F	F	F	F	F
11	The final result is in F7					

R – Run; RR – Rerun; F – Finish

Table 7. An example of inference flow

Time-3: Component ***B*** executed and provided next possible results as layer-2.

Time-4, **Time-5**: Continued the inference in the same manner.
Time-6, **Time-7**: The situation is similar as Time-3.
Time-8, **Time-9**, **Time-10**, **Time-11**: Continued the inference.
Finally, the final result is in F7 at layer-5.
Table 7 lists out the state change of components.

4.3.2 Feedback Handling

Algorithm-3 and Algorithm-4 provide forward inference with partial feedback. The procedure execution() in Algorithm-3 is to call the specific component for knowledge-based processing. When the inference engine calling execution(C_r), it passes the control to the component C_r and waits for the return of execution result. When a feedback of reasoning is considered, necessary interruption should be introduced to adjust the execution sequence. Algorithm-3 does the main job of execution control but calls Algorithm-4 (PartialRerun) to monitor feedback handling.

Considering an inference carried out in a KBS constructed by KWS, when an intelligent component failed to work out a solution as its output with its local knowledge source, an effort is expected to "bring back" the process to those field(s) or component(s) that provided the input(s) to it. It introduces a need of bidirectional inference within part of the knowledge hierarchy. This is achieved by the *Rerun* control of the KWS inference engine.

A component under *Rerun* state means it is not successful in the previous run of inference and needs 'redo' the task to provide new (better) result. The handling of rerun starts by asking new input from child component(s), according to the type of protocol between an input field of component C_r currently under *Rerun* and the output field of its corresponding child component C_c. The main algorithm is given in Algorithm-3 with further implementation details omitted.

```
Algorithm-3 (Control of execution, with $k \geq 2$ components):
1)  Get input data for all the layer-1 fields, and
            set them as of Finish;
      Set all other components and fields as of Run;
2)  While (not all the components are of Finish)
        2-1) If there are any components $C_j$, $2 \leq j \leq k$, currently under Rerun
                Then set currentFrozen : = $\min_j level(C_j)$
                Else
                  set currentFrozen := 1 + max[$level(C_1)$, $level(C_2)$, ..., $level(C_k)$];
        2-2) Check component $C_r$ ($2 \leq r \leq k$) in the execution list
                 following nondecreasing order of level:
          2-2-1) If $C_r$ is of Run and
                     $level(C_r)$ < currentFrozen and
                     all of its input field(s) are of Finish
                 Then If execution ($C_r$) /* successful */
                      Then set $C_r$ and its output field as of Finish;
                      Else set $C_r$ and its output field as of Rerun;
          2-2-2) Else If $C_r$ is of Rerun and
                     $level(C_r) \leq$ currentFrozen and
                     all of its input field(s) are of Finish
                 Then call PartialRerun($C_r$).
          /* else next component */
    /* end of while */
```

When the inference engine calling getNext(C_c), it gives a signal to ask for the next possible output from C_c. The procedure reExecution() does a similar job as execution(), but calls the component to provide next new result (if any) with the same previous input data. The KWS inference engine tries to get new input data for the component C_r currently under *Rerun*, through either getNext() or reExecution(). When the effort of getting new input data from its child component C_c is successful either through getNext() or reExecution(), all the ancestor component(s) of C_c as well as their output fields will be updated to *Run* state by calling setRunAncestor() to clean up the result of previous run. As long as one of the child component of C_r could provide new input successfully, C_r will be of *Run* again, otherwise it will remain as of *Rerun* and all its child components as well as their output fields will be set as of *Rerun*.

In case that a single child component is supporting multiple parent components, a data inconsistency should be avoided when partial feedback and rerun are considered. This consistency is guaranteed by indicating the current frozen area. A component C_c being required for a 'rerun' by one of its parent components C_r will cause a temporary 'frozen' execution to other related components. A 'frozen' execution affects two groups of components: (a) all components of *Run* state at a level equal to or higher than currentFrozen; (b) all components of *Rerun* state at a level higher than currentFrozen.

```
Algorithm-4 (Partial Rerun from $C_r$):
For each of the $h$ ($h \geq 1$) input fields of $C_r$: $\mathbf{f}_{cr1}, \mathbf{f}_{cr2}, ..., \mathbf{f}_{crh}$,
    If it is the output field of some child component $C_c$
    Then Check the protocol connection from $C_c$ to $C_r$:
        Case: multiple-to-single
            If getNext($C_c$) /* successful */
            Then set $C_c$ and its output field as of Finish;
                setRunAncestor($C_c$);
                Return;
            Else If reExecution($C_c$) /* successful */
                Then set $C_c$ and its output field as of Finish;
                    setRunAncestor($C_c$);
                    Return;
                    /* end of case multiple-to-single */
        Case: single-to-single:
        Case: single-to-multiple:
        Case: multiple-to-multiple:
            If reExecution($C_c$) /* successful */
            Then set $C_c$ and its output field as of Finish;
                setRunAncestor($C_c$);
                Return;
       /* check next input field of $C_r$ */
   /* end of 1st for */
For each of the $h$ ($h \geq 1$) input fields of $C_r$: $\mathbf{f}_{cr1}, \mathbf{f}_{cr2}, ..., \mathbf{f}_{crh}$,
    If it is a direct input from application
    Then stop processing and report failure
    Else /* it is the output field of some child component $C_c$ */
        set $C_c$ and its output field as of Rerun;
/* end of 2nd for */
```

5. Conclusions

We have introduced the KWS as a framework of development tool for developers to model and develop their customized KBS, provided the processing flow of KWS in constructing a KBS, and discussed the major sub-systems of KWS, including KWS inference engine, intelligent editor, KDL processor, and installer.

The inference in KBS constructed by KWS is a truth value flow inference (TVFI) realized at two levels simultaneously: the content level of inference that relies only on the knowledge sources stored "locally" in individual intelligent components, and the truth (confidence) level of inference that contributes to the confidence flow throughout the entire KBS. We have discussed the mechanism of TVFI as well as its implementation. The interval-valued confidence (IVC) has been adopted for the representation of imprecision and uncertainty in KBS constructed by KWS. An IVC is represented by a fuzzy number defined as a fuzzy subset of [0, 1]. Basic logic operations with IVC have been defined and their properties discussed. Based on the concepts of TVFI and IVC, confidence transfer with different types of intelligent component and corresponding interpretability has also been discussed. KWS inference engine has been explained in detail with the control algorithms of execution order of components for a forward inference with partial feedback, the management of protocols, and the handling of imprecision with TVFI and IVC.

Further effort will be put in handling knowledge imprecision with different types of intelligent processing and their integration in hybrid intelligent systems.

6. Acknowledgement

This work was supported in part by the Macao Science and Technology Development Fund under grant 048/2006/A.

7. References

L. Ding and Z. Shen (1994). Neural Network Implementation of Fuzzy Inference for Approximate Case-based Reasoning, In: *Neural and Fuzzy Systems: The Emerging Science of Intelligence and Computing*, Mitra, Sunanda.; Gupta, Madan M.; and Kraske, Wolfgang, 28-56, SPIE Press

L. Ding, H.H. Teh, P.Z. Wang. and H.C. Lui (1996). A Prolog-like inference system based on neural logic, *Fuzzy Sets and Systems*, Vol. 82, No. 2, 235-251

L. Ding and H.C. Lui (1999), A Knowledge-based Approach Applied in Intelligent Hand Written Form Processing, *Journal of Advanced Computational Intelligence*, Vol. 3, No. 3, 193-199

L. Ding (2007a). A Model of Hierarchical Knowledge Representation – Toward Knowware for Intelligent System. Journal of Advanced Computational Intelligence & Intelligent Informatics, Vol. 11, No. 10, pp. 1232-1240

L. Ding (2007b). Design and development of knowware system. Proceedings of 2nd International Conference on Innovative Computing, Information and Control (ICICIC'2007), pp. 17-17, Kumamoto, Japan.

L. Ding and S. Nadkarni (2007). Automatic Construction of Knowledge-Based System Using Knowware System. *Proceedings of 6th International Conference on Machine Learning and Cybernetics*, , pp. 789-794, Hong Kong, China

L. Ding (2008). Inference in Hybrid KBS with Interval-Valued Confidence. *Proceedings of 2008 IEEE World Congress on Computational Intelligence / 2008 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2008)*, pp.1350-1357, Hong Kong, China.

L. Ding and S.L. Lo (2008). Truth Value Flow Inference in Hybrid KBS Constructed by KWS. *Proceedings of 3rd International Conference on Innovative Computing Information and Control (ICICIC'2008)*, pp. 311-314, Dalian, China

J.-S.R. Jang, C.-T. Sun and E. Mizutani (1997). Neural-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence, Prentice Hall, NJ.

Z. Shen and L. Ding (1994). A Representation of Exponential Form on Fuzzy Logic. *Fuzzy Sets and Systems*, Vol. 68, pp.267-280

P.Z. Wang and H.M. Zhang (1993). Truth value flow inference and its mathematical theory, In: *Between Mind and Computer*, Eds. P.Z. Wang, K.F. Loe, 325-358, World Scientific, Singapore.

R.R. Yager and D.P. Filev (1994), Essentials of Fuzzy Modeling and Control, John and Wiley and Sons, Inc., NJ

L.A. Zadeh (1975), The concept of a linguistic variable ans its applicatin to approximate reasoning: Parts 1, 2 and 3, *Informtion Sciences*, 8, pp.199-249; 8, pp.301-357; 9, pp.43-80.

Machine Learning
Edited by Yagang Zhang

ISBN 978-953-307-033-9
Hard cover, 438 pages
Publisher InTech
Published online 01, February, 2010
Published in print edition February, 2010

Machine learning techniques have the potential of alleviating the complexity of knowledge acquisition. This book presents today's state and development tendencies of machine learning. It is a multi-author book. Taking into account the large amount of knowledge about machine learning and practice presented in the book, it is divided into three major parts: Introduction, Machine Learning Theory and Applications. Part I focuses on the introduction to machine learning. The author also attempts to promote a new design of thinking machines and development philosophy. Considering the growing complexity and serious difficulties of information processing in machine learning, in Part II of the book, the theoretical foundations of machine learning are considered, and they mainly include self-organizing maps (SOMs), clustering, artificial neural networks, nonlinear control, fuzzy system and knowledge-based system (KBS). Part III contains selected applications of various machine learning approaches, from flight delays, network intrusion, immune system, ship design to CT and RNA target prediction. The book will be of interest to industrial engineers and scientists as well as academics who wish to pursue machine learning. The book is intended for both graduate and postgraduate students in fields such as computer science, cybernetics, system sciences, engineering, statistics, and social sciences, and as a reference for software professionals and practitioners.