Defining and combining Multiple-valued Logics for Knowledge-based Systems

José Antonio Reyes Francesc Esteva Josep Puyol-Gruart
Artificial Intelligence Research Institute, IIIA
Spanish Council for Scientific Research, CSIC
Campus UAB. 08193 Bellaterra, Catalonia, Spain
E-mail : {reyes, esteva, puyol}@iiia.csic.es

Abstract

Multiple-valued logics are useful for dealing with uncertainty and imprecision in knowledge-based systems. In this paper, we present a tool that assist users in the declaration of such logics and in the declaration of the communication mechanism between two of these different logics by preserving inference.

Keywords : uncertain reasoning, multiple-valued logics, knowledge-based systems.

1 Introduction

The management of uncertainty and imprecision in knowledge-based systems becomes essential to model many real problems. Multiple-valued logics (MV-Logics) have been proved to be a possible way to manage them in knowledge-based systems [3; 5; 6; 10].

The aim of this work is concerned with presenting a tool which makes automatic the process of defining and communicating finite MV-Logics. This tool has been designed to be incorporated to the shell **Milord II** [7], an environment for developing knowledge-based systems, although it can be used in a more extensive framework. In section 2, we will introduce the concept of algebra of truth-values which defines a parametric family of MV-logics, as well as the syntax and the semantic permitted. Next, the extension of that algebra to an algebra of intervals of truth-values is described as a method for dealing with imprecision. Section 3 is devoted to establish the essential requirements needed to preserve inference in different ways when we communicate different logics. We give existence conditions for every requirement and identify different sorts of renaming functions between logics. In section 4, we describe the tool and its main features by means of an example. Finally, in section 5 we outline the conclusions of this work.

2 Defining MV-Logics

In this section we will see which features are necessary to define a suitable MV-Logic. Next, we will show how this process has been automated and which are the achieved results.

We consider a restricted family of finite MV-logics expressive enough to model the uncertain reasoning used in many rule-based systems [1]. Each logic is determined by a particular algebra of truth-values from this family.

An **Algebra of truth values** is a finite algebra $A_T^n = \{A_n, \mathbf{0}, \mathbf{1}, N_n, T, I_T\}$ such that :

1. The ordered set of **truth-values** A_n is a chain of n elements :

   $$\mathbf{0} = a_0 < a_1 < \ldots < a_{n-1} = \mathbf{1}$$

   where $\mathbf{0}$ and $\mathbf{1}$ are the Boolean *False* and *True* respectively, and each element a_i is a linguistic term. For example $A_5 = \{false, unlikely, may\_be, likely, true\}$.

2. The **negation** operator N_n is the unary operation defined as

   $$N_n(a_i) = a_{n-1-i} \quad (1)$$

   the only one that fulfills the following properties :

   N1 : If $a < b$ then $N_n(a) > N_n(b)$, $\forall a,b \in A_n$.

   N2 : $N_n^2 = Id.$

3. The **conjunction** operation T is any binary operation such that the following properties hold $\forall a,b,c \in A_n$:

   T1 : $T(a,b) = T(b,a)$
   T2 : $T(a,T(b,c)) = T(T(a,b),c)$
   T3 : $T(\mathbf{0},a) = \mathbf{0}$
   T4 : $T(\mathbf{1},a) = a$
   T5 : If $a \leq b$ then $T(a,c) \leq T(b,c)$ for all c

   See an example of conjunction operator in Table 1.

T_5 (x, y)	false	unlikely	may_be	likely	true
false	false	false	false	false	false
unlikely	false	unlikely	unlikely	unlikely	unlikely
may_be	false	unlikely	may_be	may_be	may_be
likely	false	unlikely	may_be	may_be	likely
true	false	unlikely	may_be	likely	true

Table 1: T_5 conjunction operation.

4. The **implication** operation I_T is defined by residuation with respect to T, i.e.

$$I_T(a,b) = Max\{c \in A_n \mid T(a,c) \leq b\} \quad (2)$$

Therefore, to establish an algebra of truth-values it is only necessary to determine the set of truth-values A_n more adequate for the concerning problem, and define a conjunction operator T necessary to combine and propagate uncertainty when making inference.

Forthwith, we introduce the syntax of our MV-Logic. The **sentences** of the language are pairs (p, V), where: p is a well formed-formula , obtained in the usual way from a denumerable set of propositional symbols and the connectives 'not' ($\neg$),'and' ($\wedge$) and 'detachment' ($\rightarrow$); and V is an interval of truth-values.

Likewise, semantic is evaluated in an algebra of truth-values The semantic interpretation is given by the followings signification :

- **Models** are defined by valuations, i.e. mappings ρ from the first components of sentences (propositional symbols) to A_n provided that :

$\rho(\neg p) = N_n(\rho(p))$

$\rho(p_1 \wedge p_2) = T(\rho(p_1), \rho(p_2))$

$\rho(p \rightarrow q) = I_T(\rho(p), \rho(q))$

- The **Satisfaction Relation** between models and sentences is defined by :

  $M_\rho \models (p, V)$ if, and only if, $\rho(p) \in V$,

  where M_ρ stands for the model defined by a valuation ρ.

Therefore, from now on, it is required that the logic systems used by applications (axioms and inference rules) must be sound with respect to the semantic above defined, in order to be considered as an appropriate MV-Logic. The inference rule of these logics is Modus Ponens, that gives from $(p \rightarrow q, V)$ and (p, V') a sentence (q, V''). In fact, modus ponens evaluates V'' from V and V' (See [9] for a theoretical study of modus ponens, and [6] for its application to expert systems).

After the definition of algebra of truth-values we extend it to an algebra of intervals of truth-values, [4]. We have some motivations to do this extension. The first one is given as consequence of imprecision, because we do not know a precise value but an interval of truth-values. A second reason is related with the modus ponens operator which requires the use of intervals to chain rules. And as we will see in the next section, it is needed to make possible mappings between different logics.

Given an algebra of truth-values $A = \langle A_n , N_n , T \rangle$, we will consider the set of intervals of A_n as:

$$Int(A_n) = \{[a,b] \mid a,b \in A_n\} \quad (3)$$

being $[a,b] = \{x \in A_n \mid a \leq x \leq b\}$.As an example, the set of intervals of a chain $A_4 = \{0 < a < b < 1\}$ is $Int(A_4) = \{[0,a], [0,b], [0,1], [a,b], [a,1], [b,1], [0,0], [a,a], [b,b], [1,1]\}$.

In the same way, we can think about extend also the algebra operators for dealing appropriately with intervals. The extensions to intervals of the above operators are : $N_n^*([a,b]) = [N_n(b), N_n(a)]$, $T^*([a,b], [c,d]) = [T(a,c), T(b,d)]$.

2.1 Conjunction Generation

The search for a conjunction operator is an important task in an algebra declaration as we can deduce from the definition of algebra of truth-values. Next we shall discuss how this question has been treated and how this process has been automated.

Throughout this work we represent the conjunction operator as a matrix, where each element is a variable. Then, for a matrix M of dimension n, we will have :

$$M = \begin{bmatrix} V_{0,0} & V_{0,1} & \cdots & V_{0,n-1} \\ V_{1,0} & V_{1,1} & \cdots & V_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ V_{n-1,0} & V_{n-1,1} & \cdots & V_{n-1,n-1} \end{bmatrix} \quad (4)$$

The mentioned algebra requirements (**T1**-**T5**) are properties that a conjunction operator must fit. We will try to assign agreeable values for the variables according to these properties.

These requirements act as constraints over the set of possible solutions. Satisfaction of each property or constraint, causes the following guidelines which influence the conjunction search generation problem :

- Commutativity allows us to consider only the set of variables $V = \{V_{i,j} , i \geq j\}$.
- Existence of absorbent and neutral elements implies to fix the values for variables $V_{0,j}$ and $V_{j,n-1}$.
- Monotonicity implies rows and columns non-decreasing.

- Associativity test is expensive in time and memory, but taking into account that $T(a_i,a_j) \leq min(a_i,a_j)$ and so, each sub-matrix

$$M_{i,i} = \begin{bmatrix} V_{0,0} & \cdots & V_{0,i} \\ \vdots & \ddots & \vdots \\ V_{i,0} & \cdots & V_{i,i} \end{bmatrix} \quad (5)$$

  is closed with respect to the conjunction operator. In this way, the full matrix will be associative if all its sub-matrices are so. This property lets us check associativity in an incremental way each time that a value is assigned to a variable of the kind of $V_{i,i}$.

However, the number of obtained operators applying these constraints is significantly high. In particular for $n > 5$ (see column A in Table 2), we obtain more than 22 matrices. This is why we can consider two new options: introducing some new desirable constraints, as well as knowing several values of the matrix to reduce the search space to be explored.

Two types of additional, and optional, constraints are discussed. First, we can think that given two truth-values different than $\mathbf{0}$ *(false)*, it does not seem reasonable that their conjunction may be $\mathbf{0}$. This is achieved by the property :

T6 Strictness: $T(a_i,a_j) \neq \mathbf{0}$, for all $i, j \neq \mathbf{0}$

In the infinite case some continuity is required. In the finite case we also want that the result of the conjunction of near values has to be not very distant. This property is defined as follows :

T7 α -Smoothness: given $\alpha \in \mathbb{N}$, T is said to satisfy α-smoothness property if $T(a_i,a_j) = a_k$ and $T(a_{i-1},a_j) = a_p$, then $k - p \leq \alpha$

Satisfaction of these new properties (**T6**,**T7**) have the following consequences in the conjunction search generation problem :

- Strictness: the generation of strict matrices of dimension n is equivalent to generate non-strict matrices of dimension n-1.
- α-Smoothness: it reduces the set of possible values for a variable taking into account column and row adjacency.

Now we consider: general constraints (**T1**-**T5**), which provide standard behavior of a conjunction operator; and additional constraints (**T6**,**T7**), that add desirable properties to a conjunction operator and decrease the number of possible solutions.

As an additional option we can pre-settled some variables values for the conjunction matrix, so it is possible to settle the more adequate behavior to the concerning problem.

Since now, we have stated the problem of define an adequate conjunction operator. In the follow, we will present how this problem had been solved.

2.2 Generation as a CSP Problem

There exist some options when we define a conjunction operator : it is possible to give a full matrix that will be checked to assure it as a conjunction operator ; it is also possible to give a partial matrix with some pre-settled values that will be whole completed ; and finally, if it is required by generating all the matrices.

The generation of operators that satisfy a determinate set of properties can be formulated as a classic CSP (Constraint Satisfaction Problem) :

- Let $V = \{V_0 , \ldots , V_{n-1}\}$ be a set of variables.
- Let $D = \{D_0 ,\ldots, D_{n-1}\}$ be a set of the corresponding discrete and finite domains (in general $D_i \neq D_j$ *if* $i \neq j$), where variables take values.
- Let $C = \{ C_{i,j} \mid 0 \leq i,j \leq m-1\}$ be a set of binary constraints. These constraints express the relationship that the different variables must hold among them for their values to be compatible.

The problem consists of finding a value assignment $\{V_{0\,|\,V_0}, \ldots , V_{n-1\,|\,V_{n-1}}\}$ such that for every $0 \leq i,j \leq m-1$, $C_{i,j}$ holds. Then, one may be interested in finding all the solutions to the problem, or just one, or the total number of solutions or if there really exists a solution.

From the concluded search influences analyzing the requirements (**T1**-**T7**) and the additional option of pre-settled some values, the CSP problem can be reformulated according to these new domains :

- Initially, we have a set of variables with settled values. This set is composed by the values concerning the variables : $V_{0,j} = \mathbf{0}$, $V_{j,n-1} = a_j$ (**T3**,**T4**) and the ones pre-settled by the user.
- We only take into account the set of variables $V = \{V_{i,j} \mid i \geq j$ with $i,j \neq 0$ and $i,j \neq n-1\}$ (**T1**).
- The initial domain for the considered variables is $D_{i,j} = \{a_0 , \ldots , a_i\}$ (**T5**).
- This initial domain can be modified according to strictness and/or α-smoothness properties (**T6**,**T7**). If strictness is applied, then the a_0 value is rejected. If α-smoothness is applied, the domain length will have a maximum of $\alpha + 1$ elements, keeping the last $\alpha + 1$ elements of the set if this number is greater.

We solved this search generation problem with a depth-first algorithm applying backtracking and look-ahead. Backtracking needs two stacks: one to keep the generated matrices and the other to keep the values that remains unassigned for each variable. The algorithm is briefly shown below. The generated solution results, taking into account **T1-T7**, are shown in Table 2.

n	*A*	*B*	*C*			*D*		
			α=1	α=2	α=3	α=1	α=2	α=3
3	2	1	2	2	2	1	1	1
4	6	2	5	6	6	2	2	2
5	22	6	13	21	22	5	6	6
6	94	22	38	78	93	13	21	22
7	451	94	118	306	422	38	78	93
8	2386	451	395	1274	2002	118	306	422
9	-	2386	1404	-	-	395	1274	2002

Table 2: Matrices per number of terms.

A **: T1-T5** ***C*** **: T1-T5** and **T7**
B **: T1-T6** ***D*** **: T1-T7**

1) *Initialization :*
   The variables with settled values are assigned and the rest will have an undefined value.
   The process order for the variables is defined.
   The stacks are empty.

2) *Variable selection :*
   The current matrix is pushed upon the matrices stack.
   A variable is chosen as the current one.
   If it is a pre-settled value, go to step 2 ; otherwise all its possible values are selected.
   If there are not more variables, go to step 5.

3) *Value selection :*
   The current variable takes on a given value.
   The rest of possible values are pushed upon the values stack..

4) *Test :*
   Check the satisfaction of all the constraints on the selected variable.
   If all these constraints are satisfied, go to step 2 ; otherwise go to step 3.

5) *Solution :*
   Each assignment passing the test is regarded as a solution.

6) *Backtracking :*
   The current matrix is popped out of the matrix stack.
   The current variable is popped out of the other stack with all unassigned values.
   If there are unassigned values left, go back to step 2 with each one.
   If the stacks are empty, then stop.

Now we know how to define a logic and how we suggest to solve and automate this process. In the next section we will see how to communicate different logics and how to preserve certain inference in this process.

3 Combining Logics

We can declare different logics by varying the set of truth-values (linguistic terms) and the conjunction operator. That depends of how the expert will deal with uncertainty in each problem.

When two different logics need to exchange information, it is necessary some mechanism of translation of the linguistic terms to make the communication between those logics compatible. At such communication, we want to preserve the inference of each logic, but as occurs in real life, it is not always possible to transmit information with absolutely precision. This loss of information can be represented with an interval, and as a consequence, we need to extend the algebra of truth-values to an algebra of intervals of truth-values. Hence we map values of an algebra into intervals of the other.

Let's take a look to which requirements we need in order to map the language of a logic into another one when they require to exchange information [2].

3.1 Mappings between MV-Logics

Let $(L, \vdash)$ and $(L', \vdash')$ be two logics, L and L' standing for the languages, $\vdash$ and $\vdash'$ for the entailment relations defined on L and L' respectively. To establish a correspondence between both logics, a *mapping* $H: L \longrightarrow L'$ is needed. Next, we will analyze some natural requirements for the mapping H with respect to the entailment systems $\vdash$ and $\vdash'$. We propose that at least one of the following three requirements should be fulfilled by the mapping H in order to ensure a consistent communication. Henceforth Γ and e will denote a set of sentences and a sentence of L respectively. A map H is said to be a *forward conservative* map when,

RQ.1. If $\Gamma \vdash e$, then $H(\Gamma) \vdash' H(e)$

For every sentence *e*, deducible from a set of sentences Γ, its corresponding sentence, *H(e)*, will also be deducible from the corresponding sentences of $H(\Gamma)$.

A map fulfilling this second requirement is said to be a *backward conservative* map :

RQ.2. If $H(\Gamma) \vdash' H(e)$, then $\Gamma \vdash e$

This is the inverse requirement of **RQ.1**. Hence, if a fact is not deducible from Γ, then its corresponding fact from $H(\Gamma)$ won't be deducible either. Nevertheless $\Gamma \vdash e$ does not imply $H(\Gamma) \vdash' H(e)$.

Conditions **RQ.1** and **RQ.2**, which are very strong, can sometimes be weakened in the uncertainty reasoning framework. Formally, this can be expressed by the third and last requirement :

RQ.3. If $H(\Gamma) \vdash' e'$, then there exists *e* such that $\Gamma \vdash e$ and $H(e) \vdash' e'$

This requirement assures that every sentence deducible from $H(\Gamma)$ must be in agreement with what can be deduced from Γ. This requirement is slightly different from **RQ.2**, in the sense that it is not necessary that *e'* is an exact translation of a deducible sentence *e* from Γ, but only something deducible from such a translation. In the framework of logics for uncertainty management, *e'* is interpreted as a *weaker* form of *e*, i.e. a sentence expressing more uncertainty than *e*. We will call it a *weak conservative* map.

Now we will consider the problem of finding inference preserving correspondences between two logics $A = <A_n , N_n , T >$ and $B = <B_m , N_n', T'>$. We are interested in mapping the entailment system $(L_A, \vdash_A)$ into the entailment system $(L_B, \vdash_B)$, by means of renaming functions between the corresponding linguistic term sets. This means that we will only consider those mappings translating sentences from L_A to L_B that just involve translations of truth-values, i.e. any mapping $H : L_A \longrightarrow L_B$ will be defined as $H(e,V) = (e,h(V))$, where *h* translates subsets of values of A_n into subsets of values of B_m, and *e* (a well-formed formula) remains invariant.

3.2 Algebra Morphisms

As it has already been noted, to establish conservative communications, it is necessary to consider what kind of relation between the uncertainty logics is required. In [2], we can find the necessary and/or sufficient conditions for a mapping $h : A_n \longrightarrow Int(B_m)$ to satisfy these requirements. From this analysis, we deduce the following relationships between the conditions needed to satisfy every requirement (see Figure 2) :

$$\textbf{RQ.1} \Leftrightarrow \left\{ \begin{array}{c} h(T(V1,V2)) \supseteq T'(h(V1),h(V2)) \\ h(N(V)) = N'(h(V)) \\ h(V1 \cap V2) = h(V1) \cap h(V2) \end{array} \right\}$$

$$\textbf{RQ.2} \Leftarrow \left\{ \begin{array}{c} h(T(V1,V2)) \subseteq T'(h(V1),h(V2)) \\ h(N(V)) = N'(h(V)) \\ h(V1) \supseteq h(V2) \Rightarrow V1 \supseteq V2 \end{array} \right\}$$

$$\Downarrow$$

$$\textbf{RQ.3} \Leftrightarrow \left\{ \begin{array}{c} h(T(V1,V2)) \subseteq T'(h(V1),h(V2)) \\ h(N(V)) = N'(h(V)) \end{array} \right\}$$

Figure 2 : Requirements conditions.

We define mappings from elements of A_n into intervals of B_m, but sometimes it is possible to find mappings that translate an element of A_n into an interval $[b_i , b_i]$ of B_m (it can be clearly consider as the element b_i). In this case, requirement **RQ.3** is satisfied and the mapping is a *morphism* between the corresponding algebras. As a particular case, if a map fulfill **RQ.1** and **RQ.2**, we have not only the morphism conditions, but a one-to-one application, that is, an injective function called *monomorphism*. But we can not always find these kind of mappings, so in the case of a map involving intervals of truth-values, we named it a *quasi-morphism*.

We are mainly interested in *monomorphisms* because they embed A_n into B_m and because they are order preserving mappings (it is equivalent to a communication without any loose of information). Then, we are interested in *morphisms*, which accomplish the algebra operations (it is a transmission of information fulfilling the required properties). Finally, because of the strong conditions *morphisms* and *monomorphisms* must satisfy, it is not always possible to find these kind of renaming functions, *quasi-morphisms* can be useful thanks to the additional freedom of map truth-values of an algebra into intervals of the other (we allow certain loose of information).

3.3 Renaming Algorithm

Before seeing the process to find these algebra morphisms, let us part from the previously well-known negation operation. For a given set of truth-values A_n, there exists only one negation N_n, and it is defined by $N_n(a_i) = a_{n-i-1}$. Then, we can partitioned this set into three subsets :

- the set of negative elements $\boldsymbol{N_n} = \{x \mid x < N_n(x)\}$
- the set of fixed elements $\boldsymbol{F_n} = \{x \mid x = N_n(x)\}$
- the set of positive elements $\boldsymbol{P_n} = \{x \mid x > N_n(x)\}$

being the subsets : $\boldsymbol{F_n} = \{a_k\}$, $\boldsymbol{N_n} = \{a_i \mid i < k\}$, $\boldsymbol{P_n} = \{a_i \mid i > k\}$, if $n = 2k+1$; and $\boldsymbol{F_n} = \varnothing$, $\boldsymbol{N_n} = \{a_i \mid i < k\}$ and $\boldsymbol{P_n} = \{a_i \mid i \geq k\}$, if $n = 2k$.

In the same way, we can do the same with a set B_m, obtaining $\boldsymbol{N^*_m}$, $\boldsymbol{F^*_m}$ and $\boldsymbol{P^*_m}$ for the case of working with intervals. Then, the renaming algorithm is as follows :

1) *Initialization* :
   Obtain the subsets $\boldsymbol{N_n}$, $\boldsymbol{F_n}$, $\boldsymbol{N^*_m}$ and $\boldsymbol{F^*_m}$.

2) *Maps Generation :*
   Generate all the maps $h_1 : \boldsymbol{N_n} \cup \boldsymbol{F_n} \longrightarrow \boldsymbol{N^*_m} \cup \boldsymbol{F^*_m}$ such that :
   a) $h_1(0) = 0$.
   b) $h_1(\boldsymbol{F_n}) \in \boldsymbol{F^*_m}$.
   c) $x \leq y$ implies $h_1(x) \not> h_1(y)$, where $x, y \in A_n$ and $h_1(x), h_1(y) \in Int(B_m)$.

3) *Map extension :*
   Extend each mapping h_1 with respect to the negation operation defining the morphism h as :

$$h(x) = \begin{cases} h_1(x), & \text{if } x \in \boldsymbol{N_n} \cup \boldsymbol{F_n}. \\ N^*_m(h_1(N_n(x))), & \text{if } x \in \boldsymbol{P_n}. \end{cases} \quad (6)$$

4) *Conjunction checking :*
   Check which ones are compatible with the conjunction operators T and T'.

5) *Renaming checking :*
   Finally, we check which maps h are morphisms, monomorphisms or quasi-morphism.

As an example, consider two logics declared with the sets of truth-values A_7 and B_5, and with the conjunction operations T_{A_7} and T_{B_5} (defined in Tables 1 and 3 respectively).

$A_7 = \{impos, few\_p, sli\_p, possib, quite\_p, very\_p, sure\}$
$B_5 = \{false, unlikely, may\_be, likely, true\}$

First, we obtain the initial subsets $\boldsymbol{N_5}, \boldsymbol{F_5}, \boldsymbol{N^*_7}$ and $\boldsymbol{F^*_7}$:

$\boldsymbol{N_5} = \{false, unlikely\}$
$\boldsymbol{F_5} = \{may\_be\}$
$\boldsymbol{N^*_7} = \{impos, few\_p, sli\_p, possib, [impos, few\_p], [impos, sli\_p], [impos, possib], [few\_p, sli\_p], [few\_p, possib], [sli\_p, possib]\}$
$\boldsymbol{F^*_7} = \{possib, [few\_p, very\_p], [sli\_p, quite\_p], [impos, sure]\}$

	impos	***few_p***	***sli_p***	***possib***	***quite_p***	***very_p***	***sure***
impos	*impos*	*impos*	*impos*	*impos*	*impos*	*impos*	*impos*
few_p	*impos*	*few_p*	*few_p*	*few_p*	*few_p*	*few_p*	*few_p*
sli_p	*impos*	*few_p*	*sli_p*	*sli_p*	*sli_p*	*sli_p*	*sli_p*
possib	*impos*	*few_p*	*sli_p*	*possib*	*possib*	*possib*	*possib*
quite_p	*impos*	*few_p*	*sli_p*	*possib*	*quite_p*	*quite_p*	*quite_p*
very_p	*impos*	*few_p*	*sli_p*	*possib*	*quite_p*	*very_p*	*very_p*
sure	*impos*	*few_p*	*sli_p*	*possib*	*quite_p*	*very_p*	*sure*

Table 3: T_{B_7} conjunction operation.

And following the above algorithm schema, we can find several mappings between both logics. There are here two mappings that are examples of those that hold the last requirement **RQ.3** :

$$\begin{cases} false \rightarrow impos \\ unlikely \rightarrow [impos, few\_p] \\ may\_be \rightarrow [few\_p, very\_p] \\ likely \rightarrow [very\_p, sure] \\ true \rightarrow sure \end{cases} \qquad \begin{cases} false \rightarrow impos \\ unlikely \rightarrow [few\_p, sli\_p] \\ may\_be \rightarrow possib \\ likely \rightarrow [quite\_p, very\_p] \\ true \rightarrow sure \end{cases}$$

When ending the generation process, the list of inference preserving mappings is presented to the user in this way :

1) First, we offer the existent monomorphisms considered as one-to-one morphisms.
2) Next, we show which morphisms are between both algebras, if any.
3) Finally, we display the list of generated quasi-morphisms.

Due to the strong conditions monomorphisms and morphisms must fit, it is not always possible to find them. However, it is very possible that the renaming generation produces a large list of quasi-morphisms, so the possibility of giving an ordered list of quasi-morphisms to aid users in their selection may be considered. Note that in the case of morphisms, all the renaming mappings have the same evaluation, hence the selection is left to the user' s criteria.

For the purpose of producing an ordered list of quasi-morphisms, we consider a weigh among the cardinal c of the set of terms which have an atomic image (that is, an interval with the form $[a_i, a_i]$) and the length L of the remainder of intervals (number of truth-values included in this interval).

Given two chains A_n and B_m, we will generate mappings between A_n and the set of intervals $Int(B_m)$. In order to obtain an evaluation for every map, we can use the following empirical function :

$$\eta = \frac{\sum_{i=1}^{n} L_i}{\sum_{i=1}^{n} c_i}, \quad 1 \leq L_i \leq m, 1 \leq c_i \leq n \quad (7)$$

where L_i is the length of the interval i and c_i is the number of points which have an atomic image. Then, for every mapping we obtain a value η, such that :

$$1 \le \eta \le \frac{(n-2)\,m+2}{2} \quad (8)$$

This method produces an ordered list of quasi-morphisms that let user to simplify the selection.

User is also able to establish the behavior that can take the generated maps defining a partial map between both algebras. This definition must satisfy the set of requirements suggested previously in order to be a negation morphism. That is, our partial map must accomplish the next points :

1) It must be a non-decreasing function.
2) If our map involves the truth-value a_0 , its image must be the truth-value b_0 . For instance, in the above example any map must fit *h(false) = impossible*. Therefore, to be a negation morphism, it is the same for the case of truth-value a_{n-1}, so it is required that *h(true) = sure.*
3) If our map implies the fixed element of the first chain A_n , we must give as its image an element belonging to the set of fixed elements F_m^* . In the above example, a map of this type must fit *h(may_be) ∈ {impos, few_p, sli_p, possib, [impos, few_p], [impos, sli_p], [impos, possib], [few_p, sli_p], [few_p, possib], [sli_p, possib]}.*

When we have defined our partial map (or total, or none), a list of renaming functions fulfilling these defined characteristics will be generated.

Now we know how to declare a MV-logic and how to establish a conservative communication between two of these logics. In the following section we will present the interactive implemented tool to define and combine MV-logics.

4 The QMORPH Tool

QMORPH [8] is a tool that allows users to define and combine MV-logics. Two different interfaces have been developed : a graphical interface for the UNIX operative system using *Tcl/Tk* packages, running under the X-Windows environment and developed on Sun machines; and a generic text mode interface performed in Common Lisp.

In the present, this tool is attached to **Milord II**, a specific expert system building environment, as an aid tool to assist experts when developing modular applications.

In this section, we illustrate the use of **QMORPH** throughout the use of a practical example in which we try to find existing morphisms between two MV-logics (as it include the particular case of a logic definition). For this purpose, we will make use of the most friendly graphical interface.

We consider the same example of section 3.3. Let us suppose we need to establish a communication between the logic $A = \langle A_5, N_5, T_5 \rangle$ (belonging to the Origin Module or O-Module) and $B = \langle B_7, N_7, T_7 \rangle$ (belonging to the Image Module or I-Module). This schema is shown if Figure 3.

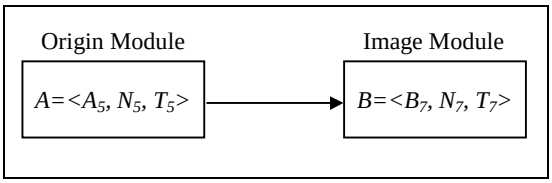

Figure 3 : Communication example.

Initially, it is mandatory to declare the truth-value sets of Origin and Image modules. Besides, user has to declare the T operation of the Origin Module for avoiding a computational explosion.

First of all, user must fix the sets of linguistic terms that settled both logics. For this purpose, it is necessary to decide how many linguistic terms have the logics (see Figure 4). Following our example , we must enter 5 and 7 terms for the Origin and Image logic respectively.

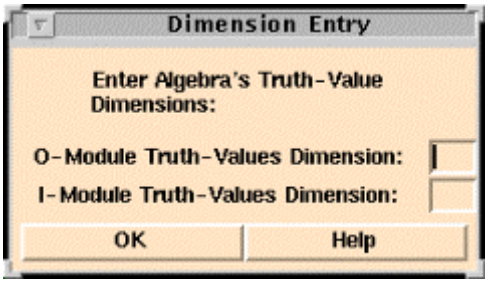

Figure 4: Entrying Algebra's Dimensions.

Next step consists in introducing the name of the terms for each logic see (Figure 5). If user does not consider necessary to name the terms, they will be generated automatically. To declare the logics A and B, we may enter here *{false, unlikely, may_be, likely, true}* and *{impos, few_p, sli_p, possib, quite_p, very_p, sure}* respectively.

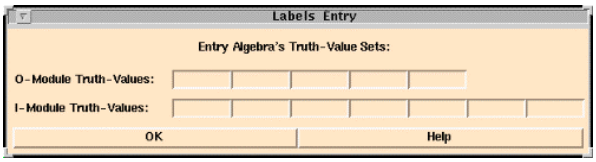

Figure 5: Entrying Truth-Values Sets.

Following, it is needed to declare the conjunction operator that *A* uses to combine and propagate the linguistic terms when making inference. User can choose a *T* function depending on the meaning he wants give to the conjunction operation. There are some matrix values that remain fixed (by properties : $T(0,a_i)=0$ and $T(1,a_i)=a_i$), so these values will be presented automatically avoiding any changes. There are some possibilities here : user can give a full operator, that will be checked to assure it as a conjunction operator; it is also possible to settle the behavior of the operator and give a partial matrix with some pre-settled values; and finally, there is the possibility of not giving any value, so all the conjunction operators would be generated. To restrict the number of possible solutions, user can apply strictness and α-smoothness constraints over the process generation. In our example (see Figure 6), we require that $T_5(unlikely, unlikely) = unlikely$.

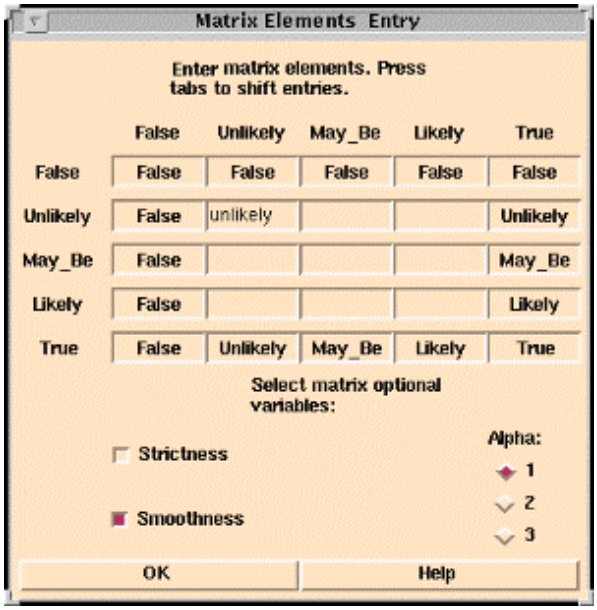

Figure 6: Entrying O-Module AND Truth-Table.

If many matrices have been generated, user must choose one of them. If user consider that there are too many matrices, it is possible to come back to redefine the matrix and/or the restrictions over it. In our example, 5 matrices has been generated (Figure 7).

To choose among the set of generated matrices, results are presented in the Matrices Reproducer (Figure 8) where we can display all the generated matrices and select the one that fits better.

Once the logic of the *A* (O-Module) has been determined, we must define the logic concerning to *B* (I-Module).

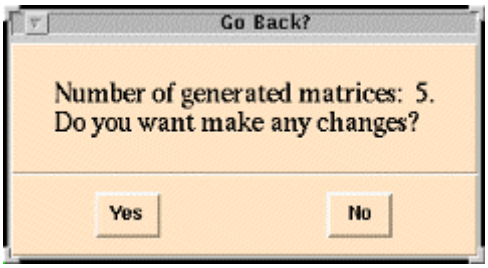

Figure 7: Number of generated matrices.

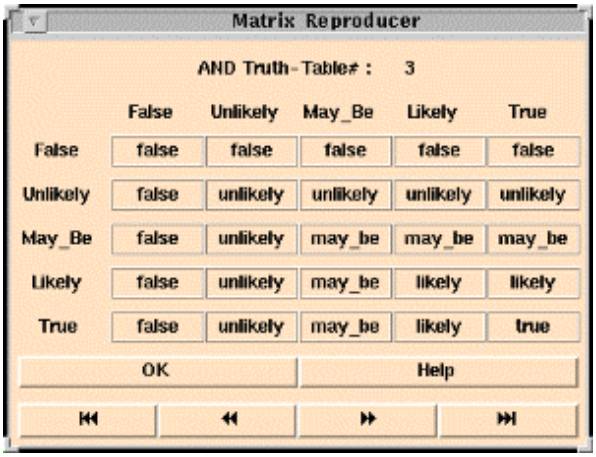

Figure 8: Browsing a conjunction operator.

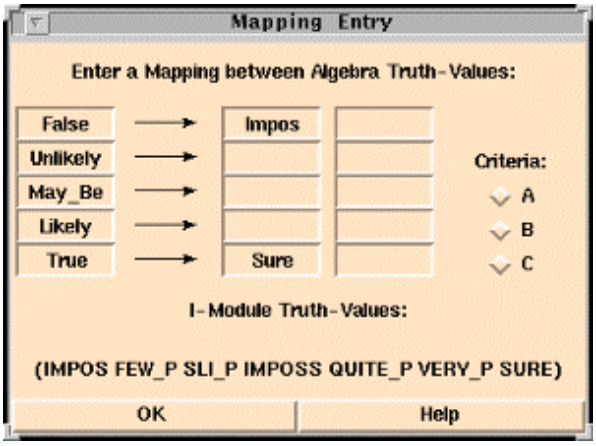

Figure 9: Defining a Renaming Mapping.

The procedure for defining an I-Module conjunction operator is exactly the same that in the case of the O-Module, but it is not necessary to choose one of them. Then, for every matrix solution, all the renaming mappings between both defined algebras will be generated. It is the possible to give a partial map between both algebras. If user desire a particular truth-value from one logic to be translated to another truth-value (or to an interval), the values of the partial map will be checked to be correct (see Figure 9). It is

permitted to choose a determinate *criteria* to be satisfied during renaming generation. There are three *criteria* : **A**, **B** and **C**, corresponding to requirements **RQ.1**, **RQ.2** and **RQ.3**, respectively.

When all mappings between the two algebras have been generated, for every I-Module operation, the possible monomorphisms and morphisms, if any, and an ordered list with all the quasi-morphisms is presented.

5 Conclusions

In this paper we have defined which are the theoretical bases that allow to deal with uncertainty by means of multiple-valued logics. We have settled how to declare a suitable MV-logic from a parametric family of algebra of truth-values, and also, which are the necessary and/or sufficient requirements when we need to establish a communication between two of these logics. This problem arises in large knowledge-based systems in which different tasks need to cooperate using uncertain reasoning, as well as in distributed systems.

We have presented the developed algorithms to aid users in the generation of conjunction operations (implemented as a classic CSP) and in the search of inference preserving mappings.

The performed tool has been designed with the aim of automating these problems, as well as assisting users and offering different alternative possibilities. Beyond its particular use within **Milord II**, it can be deployed by any other system using MV-logics.

This work focuses on the unidirectional interaction between two logics, but the more general problem of communicating various uncertainty reasoning systems is far more complex. This work along with further research will make possible this goal.

Acknowledgments

Partially supported by project MODELOGOS funded by CICYT (TIC 97-0579-C02-01) and through SMASH by by CICYT (TIC 96-1038-C04-01).

References

[1] Agustí, J.; Esteva, F.; Garcia, P.; Godo, L.; Sierra, C.; Combining multiple-valued logics in modular expert systems, *Proceedings of 7th Conference on Uncertainty in AI* Bruce d-Ambrosio et al. (eds), (1991), pages 17-25.

[2] Agustí, J.; Esteva, F.; Garcia, P.; Godo, L.; López de Mántaras, R.; Local multi-valued logics in modular expert systems, *Journal of Experimental & Theoretical AI (JETAI)*, vol. 6 n. 3 (1993), pages 303-321.

[3] Bonissone, P.; Gans, S.; Decker, K.; Rum : A layered architecture for reasoning with uncertainty, in *IJCAI'87* (1987), pages 891-898.

[4] Esteva, F.; Garcia-Calves, P.; Godo, L.; Enriched interval bilattices : An approach to deal with uncertainty and imprecision, *International Journal on Uncertainty, Fuzzyness and Knowledge-Based Systems*, 1-2 (1994).

[5] Godo, L.; López de Mántaras, R; Sierra, C.; Verdaguer, A.; Milord : The architecture and management of linguistically expressed uncertainty, *International Journal of Intelligent Systems*, Vol. 4 n. 4 (1989), pages 471-501.

[6] López de Mántaras, R.; *Approximate Reasoning Models*, Ellis Horwood Series in Artificial Intelligence, 1990.

[7] Puyol, J.; *MILORD II: A language for knowledge-Based Systems*, Vol. 1 of Monografies del IIIA, IIIA-CSIC, 1996.

[8] Reyes, J. A.; QMORPH : A tool to define and combine local logics in Milord II, *Mst. Thesis, Universitat Autònoma de Barcelona*, (1997).

[9] Trillas, E.; Valverde, L.; On mode and implication in approximate reasoning, in *Gupa et al. (eds.)*: Approximate Reasoning in expert systems, (1985), pages 157-166.

[10] Turner, R.; *Logics for Artificial Intelligence*, Ellis Horwood Series in Artificial Intelligence, 1984.