3-2003

Efficient Decomposition of Large Fuzzy Functions and Relations

Paul Burkey
Portland State University

Marek Perkowski
Portland State University

Citation Details

Published as, Burkey, Paul, and Marek Perkowski. "Efficient Decomposition of Large Fuzzy Functions and Relations." In Proceedings of International Conference on Fuzzy Information Processing. Theories and Applications, 2003, pp. 145-154.

Efficient Decomposition of Large Fuzzy Functions and Relations

Paul Burkey +, Marek Perkowski

+ Portland State University, Dept. Electrical Engineering, Portland, Oregon 97207, Tel. 503-725-5411, Fax (503) 725-4882, Email: paburkey@ee.pdx.edu, Dept. Electr.l Engn. and Computer Science, Korea Advanced Institute of Science and Technology (KAIST), 373-1 Guseong-dong, Yuseong-gu, Daejeon, 305-701, Korea.

Abstract
This paper presents a new approach to decomposition of fuzzy functions. A tutorial background on fuzzy logic representations is first given to emphasize next the simplicity and generality of this new approach. Ashenhurst-like decomposition of fuzzy functions was discussed in [3] but it was not suitable for programming and was not programmed. In our approach, fuzzy functions are converted to multiple-valued functions and decomposed using an mv decomposer. Then the decomposed multiple-valued functions are converted back to fuzzy functions. This approach allows for Curtis-like decompositions with arbitrary number of intermediate fuzzy variables, that have been not presented for fuzzy functions by the previous authors. Extension of the method to fuzzy relations is also shown. The new approach is suitable for Machine Learning.

1. INTRODUCTION

Decomposition of a function is a process of creating an equivalent composition of other, simpler functions. For example, if x and y are sets of variables and $F(x, y) = H(G(x), y)$, then the term to the right is a composition of functions that is equivalent to the function F. Thus, the complexity of F is reduced by representing function F in terms of functions G and H. The formulation of functional decomposition is very simple, but it is a very complex problem to solve when large data have to be dealt with in order to find the composing functions of the smallest total complexity. One problem is in determining how to group the input variables x and y for functions G and H. This process of selecting the input variables to G and H is called *variable partitioning* or variable grouping. The input variables going to G are called the bound set and those going to H are called the *free set*. The need for the introduction of fuzzy functions and multiple-valued functions is to extend the domain of binary functions. The world can not always be conveniently represented in binary terms so the concepts of fuzzy-valued, multiple-valued, and continuous-valued functions have been introduced. They find many applications other than circuit design, primarily Artificial Intelligence (AI), Machine Learning (ML), Fault Diagnosis, industrial control, Data Mining, Robotics, Knowledge Discovery from Data Bases (KDD), Multi-Objective Optimization, and many others. Binary functions have only two values, either 0 or 1, while a multiple-valued function can have many values. In fuzzy functions, normally defined, the values are continuous in the range from 0 to 1. Decomposing fuzzy logic functions is a difficult problem because fuzzy logic is non-disjoint. In other words, various uses of fuzzy variables cannot be separated based on their complemented or non-complemented values.

The definition, operations, identities and differences between the fuzzy logic and binary logic will be explained in the section 2. Fuzzy maps and S-maps are then introduced. Next the steps to perform fuzzy logic decomposition using fuzzy maps as in the Kandel and Francioni method [3] will be briefly mentioned and some difficulties pointed out (section 3). Then the new approach based on converting a fuzzy function to a multiple-valued function [1, 2] and decomposing the multiple-valued function will be explained (sections 4 and 5). We previously developed several decomposers and made use of them to decompose multiple-valued functions and relations [12, 13, 14]. These programs allow to deal with hundreds of variables, tens of thousands of terms, and solve efficiently difficult real-life problems from ML and KDD. The presented approach converts a multi-output fuzzy function to a multi-output three-valued function to be given as an input data to one of our decomposers. Finally, the method of converting the multiple-valued functions back to a fuzzy function will be explained in section 7 in order to prove that the network is a correct decomposition of the initial function. Section 7 presents extension of this method to fuzzy relations. Experimental results are in section 8, and section 9 concludes the paper.

1.1. Background on Fuzzy Logic

A fuzzy set, defined as A, is a subset of the universe f discourse U, where A is characterized by a membership function μ_A (x). The membership function μ_A (x) is associated with each point in U and is the "grade of membership" in A. The membership function μ_A (x) is assumed to range in the interval [0, 1], 0 corresponding to non-membership and 1 corresponding to full membership. The ordered pairs form the set $\{(x, \mu_A (x))\}$ to represent he fuzzy set member and the grade of membership [4].

A.1. Operations: The fuzzy set operations [5] are defined as follows. Intersection operation of two fuzzy sets uses the symbols: $\cap$, $\wedge$, *, AND, or min. Union operation of two fuzzy sets uses the symbols $\cup$, $\vee$, +, OR, or max. Equality of two sets is defined as $A = B \leftrightarrow \mu_A (x) = \mu_B (x)$ for all $x \in X$. Containment of two sets is defined as $A \subseteq B \leftrightarrow \mu_A (x) \leq \mu_B (x)$ for all $x \in X$. Complement of a set A is defined as A' where $\mu_{A'} (x) = 1 - \mu_A (x)$ for all $x \in X$. We will use also notation with bar on top of the argument to denote negation. Intersection of two sets is defined as $A \cap B$ where $\mu_{A \cap B} (x) = \min\{ \mu_A (x) , \mu_B (x) \}$ for all $x \in X$ where $C \subseteq A$, $C \subseteq B$ then $C \subseteq A \cap B$. Union of two sets is defined as $A \cup B$ where $\mu_{A \cup B} (x) = \max\{ \mu_A (x) , \mu_B (x) \}$ for all $x \in X$ where $D \supseteq A$, $D \supseteq B$ then $D \supseteq A \cup B$. Example 1: An example of fuzzy operations: Let X = {1,2,3,4} and consider the following fuzzy sets A and B: A = {(3, 0.8), (5, 1), (2, 0.6)} and B = {(3, 0.7), (4, 1), (2, 0.5)}. Then $A \cap B$ = {(3, 0.7), (2, 0.5)}, $A \cup B$ = {(3, 0.8), (4, 1), (5, 1), (2, 0.6)}, A' = {(1, 1), (2, 0.4), (3, 0.2), (4, 1), (5, 0)}.

A.2. Identities: The identities use fuzzy variables which are the same as elements in a fuzzy set. The definition of an element in a fuzzy set, $\{(x, \mu_A (x))\}$, is the same as a fuzzy variable x and this form will be used in the remainder of the paper. Fuzzy functions are made up of fuzzy variables. The identities for fuzzy algebra [6] are: Idempotency: X + X = X* X = X . Commutativity: X + Y = Y + X, X * Y = Y * X . Associativity: (X + Y) + Z = X + (Y + Z) , (X * Y) * Z = X * (Y * Z) . Absorption: X + (X * Y) = X, X * (X + Y) = X . Distributivity: X + (Y * Z) = (X + Y) * (X + Z) , X * (Y + Z) = (X * Y) + (X * Z) . Complement: (X') ' = X . DeMorgan's Laws: (X + Y)' = X' * Y' , (X * Y)' = X' + Y' .

A.3. Transformations: Some transformations of fuzzy sets: x' b + xb = (x + x') b ≠ b , xb + xx'b = xb(1 + x) = xb , x'b + xx'b = x'b (1 + x) = x'b , a + xa = a (1 + x) = a , a + x'a = a (1 + x') = a , a + xx'a = a , a + 0 = a , x * 0 = x , x + 1 = x , x * 1 = x . Example 2: a + xa + x' b + xx' b = a(1+x) + x'b(1+x) = a + x' b
Example 3: a + xa + x'a + xx'a = a (1 + x + x' + xx') = a

1.2. **Differences between Boolean Logic and Fuzzy Logic.**

In Boolean logic the value of a variable and its inverse are always disjoint *(X * X' = 0)* and *(X + X' = 1),* because the values are either zero or one. However, in fuzzy logic the membership functions can be either disjoint or non-disjoint. The membership function is determined by the grade of membership and can be any value in the interval [0, 1]. Fuzzy membership functions can be any function that can be realized in the interval from zero to one. For simplicity, the term "grade of membership" of a variable in a set will be replaced by the term "fuzzy variable". An example of a fuzzy non-linear membership function *X* is shown in Fig. 1a with its inverse membership function shown in Fig. 1b. The fuzzy intersection of variables X and its complement X' is not empty, or is not always equal to zero because the membership functions are non-disjoint. From the membership functions Figures 2a and 2b the intersection of fuzzy variable X and its complement X' is shown in Fig. 3a. From the membership functions Figures 2a and 2b the union of fuzzy variable X and its complement X' is shown in Fig. 3b.

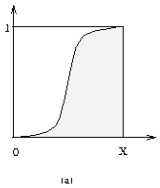
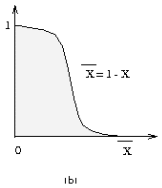


Figure 1. Non-linear membership function and its inverse.



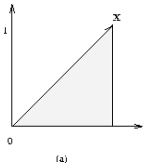
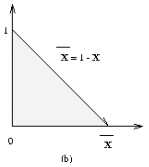
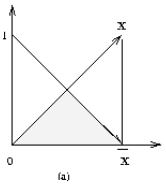


Figure 2. Linear membership function X and inverse





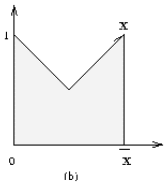


Figure 3. (a) Intersection X * X' ≠ 0. (b) Union X + X' ≠ 1.

2. GRAPHICAL REPRESENTATIONS OF FUZZY FUNCTIONS FOR DECOMPOSITION.

In Karnaugh maps [8] the symbols "1", "0" and "–" (used to denote a don't care) are used to describe minterms and cubes of a binary function and each cell corresponds to a minterm. In contrast, in fuzzy maps the whole terms are represented as cells in the map. Since there is only a finite number of unique terms in a fuzzy function, a symbol I can be used to show if a term is present [7].

Fuzzy Maps. As presented by Schwede and Kandel [9], the fuzzy map may be regarded as an extension of the Veitch diagram [10], which forms the basis for the Karnaugh map. Fuzzy maps pictorially describe the set of all fuzzy implicants which represent a fuzzy function. A K-map of n variables can be represented by 2^n areas (cells) in the map corresponding to care minterms (values 1 and 0) and don't care minterms (values -). A fuzzy map of n variables can be represented by 4^n areas (cells) in the map. The symbol I is used in the map to represent a term existing in the fuzzy function, $F(x_1, x_2, \ldots, x_n)$. For two variable fuzzy map_ the columns are labeled $x_1\ x_1'\ x_1\ x_1'\ 1$ and the rows are labeled $x_2\ x_2'\ x_2\ x_2'\ 1$, as shown in Fig. 4. The column and row headings are conventionally replaced with quarternary numbers representing the binary headings. There are four combinations for each variable x_i , i = 1,2,…,n variables, to be represented in the headings of the rows and columns, as shown in Fig. 5a.

1. This heading is vacuous in x_i . The pair $x_i x'_i$ is denoted by 00 and is represented by 0
2. This heading includes x'_i but not x_i . The pair $x_i x'_i$ is denoted by 01 and represented by 1.
3. This heading includes x_i but not x'_i . The pair $x_i x'_i$ is denoted by 10 and represented by 2.
4. This heading includes x_i and x'_i . The pair $x_i x'_i$ is denoted by 11 and represented by 3.

The construction of fuzzy maps of max, OR, +, as union, and min, AND, *, as intersection, is shown in Fig. 5. The place where I is to be placed is easy to determine. The function of union $f(X_1, X_2) = X_1 + X_2$ is shown in Fig. 5a with the X_1 term that is denoted by the I in the last row because X_2 is vacuous in this term, while the X_2 term is denoted by I in the second row because X_1 is vacuous. In Fig. 5b the function intersection $f(X_1, X_2) = X_1 * X_2$ is shown by placing an I in the column X_1 and row X_2 . Fuzzy map representation has important properties which distinguish them from Boolean maps. As in Boolean maps one can form a cube to reduce the function by circling the ones. In fuzzy maps, the placement of I can show a reduction of the fuzzy logic function. Also another placement of I can show the expansion of the fuzzy logic function (see Fig. 4) . Functions from Fig. 4a and Fig. 4b are equivalent, but have symbols I placed differently. Reduction and transforming to a canonical form of a function correspond then to moving symbols I across the map.

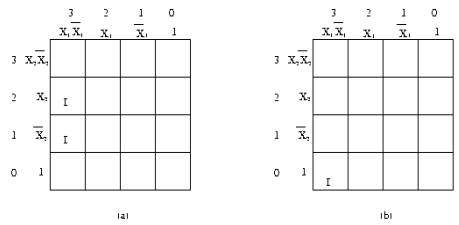


Figure 4. Fuzzy Maps with n=2, (a) $f(x_1,x_2) = x_1\ x'_1\ x_2 + x_1\ x'_1\ x'_2$; (b) $x_1\ x'_1$

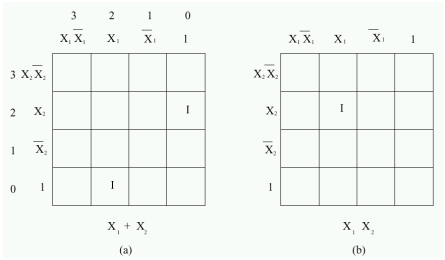


Figure 5. Max and Min representations using fuzzy maps for two variables (a) MAX, (b) MIN

The lattice of two fuzzy variables is shown in Fig. 7a with the most reduced terms on top. The lattice shows the relationship of all the possible terms. The lattice also shows which two terms can be reduced to a single term. In the corresponding fuzzy map of two variables shown in Fig. 7b the highest level is 1 and the lowest is 5. This fuzzy map shows the level in the lattice

The Subsumtion Rule. The subsume rule is a way to reduce a fuzzy logic function, because rules ($X * X' = 0$) and ($X + X' = 1$) are not valid for fuzzy logic. The subsumtion rule is based on the fact that $X * X' \leq 0.5$ and $X + X' \geq 0.5$. Also on the transform $a + xa = a\ (1+x) = a$. $\alpha\ x_i\ x'_i\ \beta + \alpha'\ x_i\ x'_i\ \beta = x_i\ x'_i\ \beta$ where α and β can be one or more than one variable [9]. Fig. 7.explains the subsumtion operation on maps of two fuzzy variables, x_i and , x'_i .

In each map, a cell marked with I denotes a term, and the cells marked with i denote all the cells subsumed by cell I. Subsumtion operations for all possible product terms of two variables are shown in Fig. 7.

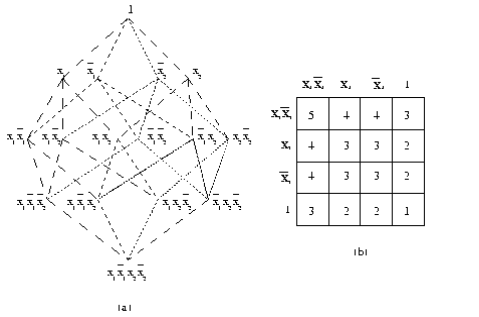


Figure 6. Representations for two variable functions (a) Lattice, (b) Level Map

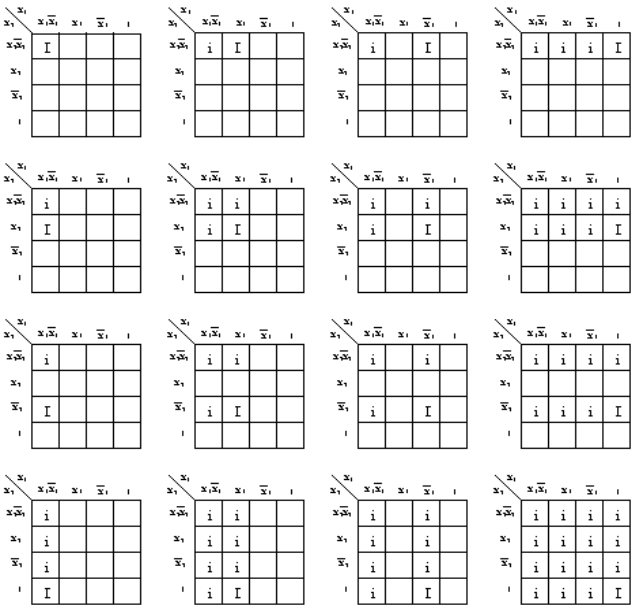


Figure 7. Subsumtion operation for all terms of two variables

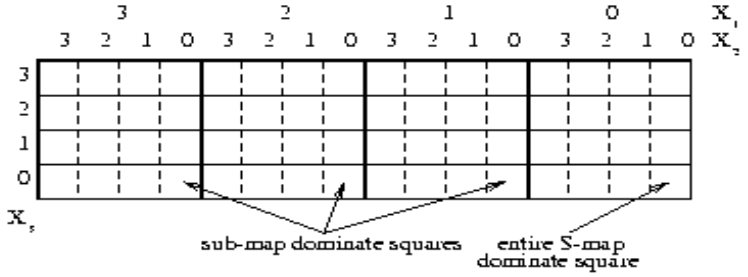


Figure 8. S-map for n =3

S-Maps. S-maps are another way to arrange two-variable fuzzy maps for n variables [9]. To construct an n-variable S-map, whole one or two variable fuzzy maps are treated as though they were squares of an S-map on n-1 or n-2 variables. This method is just iterated for n variable S-maps. These subsets of the logical space are called sub-maps and are a very important feature of S-maps. As in fuzzy maps, the binary headings for the columns and rows are converted to a quarternary representation as shown in Fig. 8. The sub-map boundaries are indicated by the vertical solid lines. The same manipulations used on a two-variable map can be used on an n-variable S-map. On S-maps, entire sub-map sized patterns behave as single cells in two-variable map [3]. Both fuzzy maps and S-maps have been used in the past to decompose fuzzy switching functions: Fuzzy map is used to _find if a decomposition exists. S-map is used to determine the decomposition, or to calculate the predecessor function G and successor function H.

3. KANDEL'S AND FRANCIONI'S APPROACH TO FUZZY LOGIC DECOMPOSITION.

The approach of Kandel and Francioni [3] was based on graphical representations and required reducing functions to canonical forms. Thus, it was quite difficult to program, which was perhaps the reason that it was not implemented in Francioni's Ph. D. Thesis. We are not aware of any other decomposer of fuzzy functions. Below we will briefly present reduction of fuzzy functions to canonical forms to make this paper a simple tutorial on fuzzy logic and also to emphasize the difficulties of Kandel's/Francioni's approach.

Function Form Needed to Decompose a Fuzzy Logic Function in [3]. As a standard, a fuzzy logic function needs to be in a canonical sum-of-products form as the input to decomposition or other minimization procedure. The steps to get a fuzzy logic function into the canonical form are the following, and will be explained next [11]:

1. Represent the fuzzy logic function in sum-of-products form.
2. Represent the fuzzy logic function in a canonical form.

This is done using the identities and transforms of fuzzy logic.

Example 4. When the function is not in sum-of-products form it is transformed to sum-of-products form: $F = X_1 (X_1 X_2 X'_2)' (X_1 X_2 X_3)' = X_1 (X'_1 + X'_2 + X_2) (X'_1 + X'_2 + X'_3) = X_1 X'_1 + X_1 X'_1 X'_2 + X_1 X'_1 X'_3 + X_1 X'_2 + X_1 X'_2 X'_3 + X_1 X'_1 X'_2 + X_1 X_2 X'_2 + X_1 X_2 X'_3 = X_1 X'_1 + X_1 X'_1 X'_2 + X_1 X'_1 X'_3 + X_1 X'_2 + X_1 X'_2 X'_3 + X_1 X_2 X'_2 + X_1 X_2 X'_3$.

Eliminate the terms which can be subsumed by other terms. After changing the fuzzy logic function into the sum-of-products form, the function is not reduced to its simplest form. The function now needs to be reduced to its simplest form to be canonical. This is done using the subsume Rule, as shown in Fig. 7. **Example 5:** The function $F(x_1, x_2) = x_2 x'_2 + x'_1 x_2 + x_1 x'_2 + x_1 x'_1 x'_2$ is equivalent to $F(x_1, x_2) = x'_1 x_2 + x_1 x'_2$ as shown in Fig. 9.

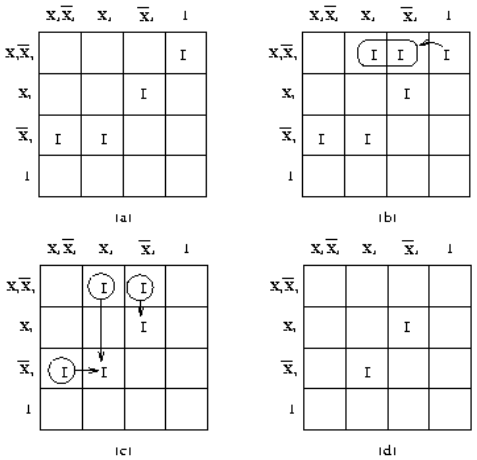


Figure 9. Using fuzzy maps to find the canonical form of $f(x_1, x_2)$

Problems with using Kandel's and Francioni's method of Fuzzy logic decomposition.

Their decomposition method, [3], uses fuzzy maps to determine if a fuzzy decomposition exists and then uses S-maps for the fuzzy decomposition. Theorems, definitions, and a table are used to tell if the function in a fuzzy map or a S-map is decomposable and how to perform the decomposition. Special patterns must be recognized in the maps. Their method was not implemented as a computer program, nor its correctness was verified, and it is difficult to use both fuzzy maps and fuzzy S-maps concurrently. Conceptually and didactically the many theorems from [3] and definitions are hard to explain because they are not linked in any way to the well-known concepts of Ashenhurst/Curtis decomposition. Besides, the method is difficult to extend to an arbitrary number of variables and to Curtis-like decompositions. In the sequel we will explain the steps of converting any fuzzy function to a functional form that can be decomposed using multiple-valued decomposition approaches developed recently [12,13,14]. The next section will show how a fuzzy function in a sum-of-products form can be converted to a multiple-valued function, decomposed, and next converted back into a multiple-valued fuzzy function.

4. FUZZY FUNCTION TO MULTIPLE-VALUED FUNCTION CONVERSION

The procedure to convert a fuzzy function to the multiple-valued (MV) function is the following:

4.1. A fuzzy logic function needs to be in a sum-of-products form.

4.2. The new map for the MV function needs to be of dimension equal to the number of variables in the fuzzy function. Every variable X_i in the map will have 3 values. The value $X_i = 0$ is used in the map where variable's complement is present in the term. The value of 1 is used when the variable and its complement are present in the term, and the value of 2 is used in the map where the variable is present in the term.

4.3. For every product term of the fuzzy function, convert all variables to ternary form and perform the MIN operations on them.

4.4. After multiple-valued map for each product term is created, the cells which are covered by these products are MAX-ed together to create the function's multiple-valued map.

Fig. 10 explains the mapping between the fuzzy terms and terms in the MV map. A whole row or column of cells corresponds to a single variable. For instance, all cells in column 1 are for $x_2\ x'_2$. The next example shows how converting a fuzzy function to MV function reduces the function to a canonical form, as example 5 showed for fuzzy maps.

Example 6. We use the same function as in example 5 to show how to convert a fuzzy function into an MV function. This example shows that converting a fuzzy function into a MV function reduces to a canonical form because as shown in Fig. 12 and in Fig. 13 the results are the same.

The next section shows an example of taking a fuzzy function and converting it into a MV function to be decomposed and then converted back into two fuzzy functions.

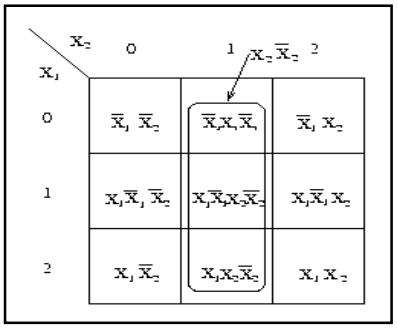


Figure 10. Conversion of fuzzy terms to Multiple-valued terms on variables x_1 and x_2 . For instance, fuzzy term = $x_1\ x'_1 + x_2\ x'_2$ is converted to $X_1^{\ 1}\ X_2^{\ 1}$ and fuzzy term $x_1\ x'_1\ x'_2$ is converted to $X_1^{\ 1}\ X_2^{\ 2}$

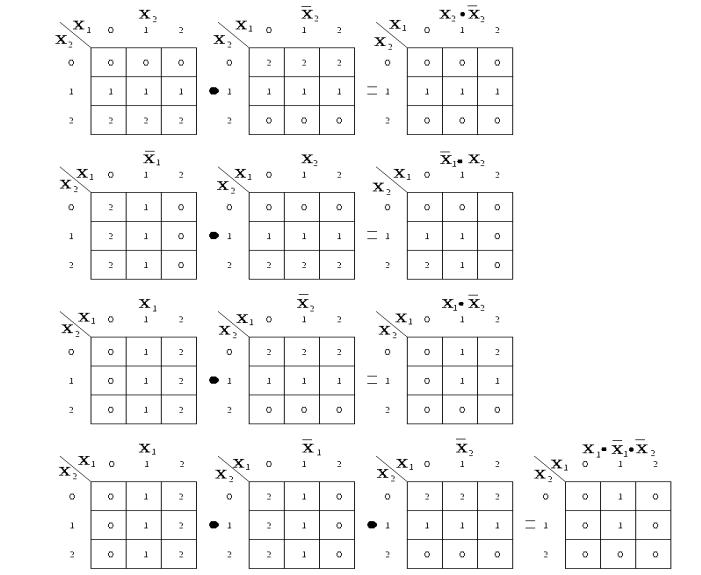


Figure 11. Conversion of a Fuzzy Function from Example 6 in non-canonical form to a Multiple-Valued Function.

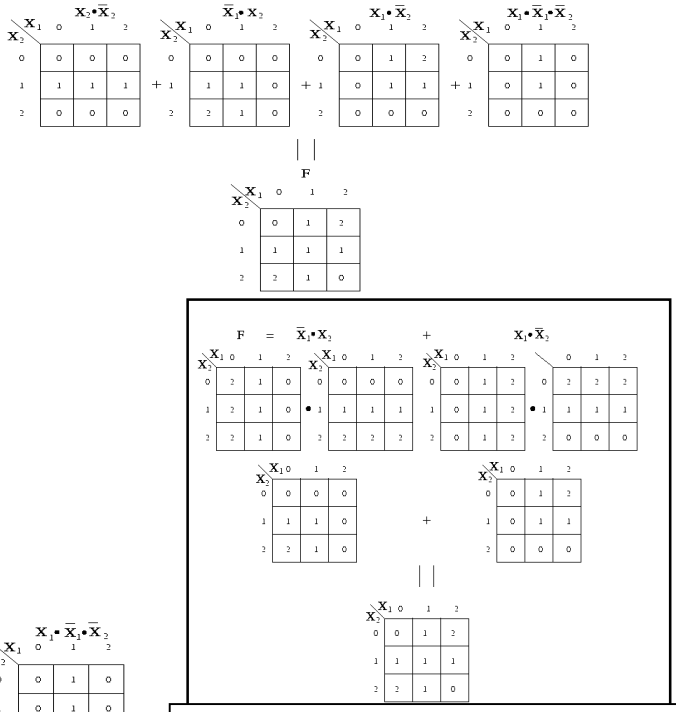


Figure 12. Conversion of a Fuzzy Function from Example 6 in a canonical form to a Multiple-Valued Function. Observe the same result as in Fig. 11.

5. DECOMPOSITION OF MULTIPLE-VALUED FUNCTIONS.

Example 7. $F(x, y, z) = yz + x'\ y'\ zz' + xz$. Fig. 13 shows the conversion of this three-variable fuzzy function to a ternary MV function of three ternary variables. The first map corresponds to xz , the second map corresponds to

x' y' z z' and the third map corresponds to yz_ The map on bottom is the maximum of the three maps above and it represents the ternary function F(X, Y.Z) to be decomposed. Let us observe that for all possible cofactors X=i, Y=j , I, j = 0,1,2, the characteristic patterns 010, 011 and 012 exist. For instance, pattern 011 exists for X=0 and Y=1; X=1 and Y=1; X=1 and Y=0 . Pattern 010 exists only for X=0 and Y=0. Pattern 012 exists for all other combinations of X and Y values. Thus from [13,14] the function has three patterns for the bound set {X, Y} and is Ashenhurst decomposable, which means that only one intermediate (ternary) signal G is needed. The input function table is taken from the result of the Fig. 13. The tables of functions G and H are the result of the MV decomposition. Such decomposed functions can be obtained using any of the two developed by us decomposers [13,14], or any other general-purpose or ternary decomposer (Table 1).

6. CONVERTING THE MULTIPLE-VALUED FUNCTION TO THE FUZZY FUNCTION

The initial fuzzy function is converted to the multiple valued function and then decomposed to several interconnected multiple-valued functions called blocks. After completing the iterative multi level decomposition process of multi valued functions to non-decomposable blocks, [13,14], the block functions need to be converted back to fuzzy functions. The procedure to convert a multiple valued block function to a fuzzy function is:

6.1. Use multiple valued minimization to minimize the function option.

6.2. Convert the multiple valued product terms back into fuzzy product terms where each variable value $X_i = 1$ is converted into a variable and its complement $x_i\ x'_i$. Each variable value $X_i = 2$ is converted into the variable x_i and each variable value $X_i = 0$ is converted to the variable's complement x'_i .

The results of the decomposition process, functions G and H are shown in Fig. 15, a, b, respectively, as MV maps. Fuzzy terms Gz G'zz' and zz' of H are shown. Two solutions are obtained, G(x, y) = x + y, H(x, y) = Gz + zz' (Fig. 14c) and G(x, y) = x + y, H(x, y) = Gz + G' zz' (Fig. 14d). The correctness of these two decompositions can be verified by a reader by drawing all intermediate MV maps (as in Example 6) that create functions G and H , composing functions G and H back to F , and converting from ternary to fuzzy. In this case functions G and H are not decomposed further but in general these functions can be decomposed thus creating a tree or a Directed Acyclic graph of decomposed fuzzy blocks.

Input Function

X	Y	Z	F
0	0	0	0
0	0	1	1
0	0	2	0
1	0	0	0
1	0	1	1
1	0	2	1
2	0	0	0
2	0	1	1
2	0	2	2
0	1	0	0
0	1	1	1
0	1	2	1
1	1	0	0
1	1	1	1
1	1	2	1
2	1	0	0
2	1	1	1
2	1	2	2
0	2	0	0
0	2	1	1
0	2	2	2
1	2	0	0
1	2	1	1
1	2	2	2
2	2	0	0
2	2	1	1
2	2	2	2

(a)

G Function

X	Y	G
0	1	1
1	0	1
2	0	2
0	0	0
1	2	2
2	2	2
0	2	2
1	1	1
2	1	2

(b)

H Function

Z	G	F
2	2	2
1	2	1
0	2	0
2	1	1
1	1	1
0	1	0
2	0	0
1	0	1
0	0	0

(c)

Table 1

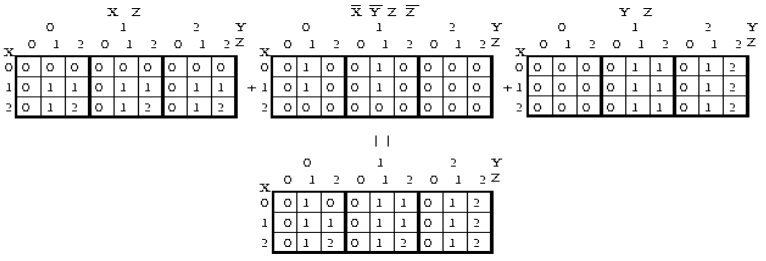


Figure 13. Conversion of three-variable fuzzy function F(x,y,z)=xz+x'y'zz'+yz from Example 7 to MV function.

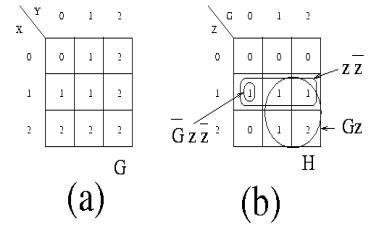
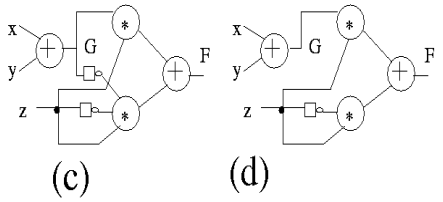


Figure 14. (a) Resulting MV Map of function G, (b) Resulting MV Map of function H, (c) First variant of decomposition, (d) Second variant of decomposition



7. DECOMPOSITION OF FUZZY RELATIONS.

Multi valued relation is introduced in [13] as a table in which for certain combination of input variables values one of several specified output values can be selected. For instance, in Figure 15g in cell for $z = 1$, $G = 0$ there are two values, 0 and 1. It means that any ternary value other than value 2 can be taken for this combination of input variable values. This is called a generalized don't care and it generalizes a standard don't care concept where any set of values of a given output is allowed for given input combination. Thus, the generalized don't cares of a ternary signal are: {0,1}, {1,2} and {0,2}. The standard don't care is {0,1,2}. Let us observe that the generalized and standard don't cares correspond to the following values in fuzzy logic: $\{0,1\} = x'_i$ or $x_i x'_i$ (when an undecided shape is between the one from Fig.2b and the one from Fig. 3a). $\{1,2\} = x_i x'_i$ or x_i (when an undecided shape is between the one from Fig. 3a and the one from Fig. 2a). $\{0,2\}$ x'_i or x_i (when an undecided shape is between the one from Fig. 2b and the one from Fig. 2a). {0,1,2} when the shape of x_i is irrelevant. There are several ways to specify the initial fuzzy relations A graphical method is illustrated in Figure 15a. The OR relations among groups of terms denote that the choice of any of the groups of terms pointed by the two arrows originating from word OR can be made. Thus the function from Fig. 15 is specified by the expression*: F(x, y, z) = yz CHOICE-OF[x' y' z z ' OR (z z'x x' y' + z z' y y' x')] + xz.* In general, a fuzzy relation can be specified by an arbitrary multi-level decision unate function on variables G_I, each of these variables denoting Max of terms for a sum-of-products form of fuzzy relation. Such unate function uses functors AND and OR and variables G I corresponding to Max groups of terms. The above fuzzy relation is specified by the unate decision function: A AND B AND (C OR D) = (A AND B AND C) OR (A AND B AND D) where: A = *yz* , B = *xz*, C = *x' y' z z'* , D = *(z z' x x' y' + z z' y y' x').* Thus, every fuzzy relation corresponds to a set of sum-of-products fuzzy functions among whichwe can freely choose.

Example 8. Given is a fuzzy relation $F_r (x, y, z) = yz +$ CHOICE-OF[x' y' z z' OR (z z' x x' y' + z z' y y' x')] + *xz* , illustrated also in the map from Figure 16a. This is modification of Example in which more choices of fuzzy terms are given to the optimization tool. We specify that the tool has a freedom of choice between the groups of terms C = *x' y' z z'* or D = *(z z' x x' y' + z z' y y' x'),* which ever simplifies the final solution more.

For this fuzzy relation the map of ternary relation from Figure 15e is created by the operation of Maxing the ternary maps of functions xz (Fig. 15b), *yz* (Fig. 15d), and the map of the ternary relation corresponding to fuzzy relation [CHOICE-OF *x' y' z z'* OR (*z z' x x' y' + z z' y y' x'*)] (Fig. 15c). Observe that there are two entries, 0 and 1 in the cell $x = 0, y = 1, z = 1$ in Fig. 15e; this cell is called a generalized don t care and thus Fig. 15 stores a ternary relation, not a ternary function. The characteristic patterns found for Ashenhurst-like decomposition are encircled in Fig. 15e. Other patterns found are 011 and 0(0,1)0. The last pattern corresponds to either pattern 000 or to pattern 010. Thus, in any case there are three patterns, and the decomposition exists. Ternary function G after decomposition is shown in Figure 15f and ternary relation *H* is shown in Figure 15g . In general, both *G* and *H* can be relations in our approach, so our decomposition decomposes a relation to relations. Interestingly, sometimes also a function can be decomposed to relations. As we see, there is a choice of 0 and 1 in cell z=1, $G =$ 0 in Figure 15g. Choice of value 0 (Fig. 15g, $H = GZ$) leads to the simpler solution from Figure 15h. Alternately, the choice of value in Fig. 15g leads to the more complex solution from Figure 15i, which was found earlier in Example 7, when function *F* was assumed instead of relation F_r . Transforming, when possible, a fuzzy function to a fuzzy relation, has thus a similar effect as replacing some of cares of a function by don't cares - it can be better minimized.

8. EXPERIMENTAL RESULTS.

We decomposed correctly all functions from [3,6] and from other papers on fuzzy logic and the computer times were negligible. The decomposer from can be set to any fixed number of values in all intermediate signals, so it is set to the value of three for ternary logic that corresponds to fuzzy logic. The decomposer from [14] decomposes to arbitrary-valued intermediate signals, in order to maximally decrease the total circuit's complexity and decrease the recognition error. It requires then encoding the signals that have more than three values to ternary vectors which is done by hand. For instance an intermediate signal with values 0, 1, 2 and 3 is encoded to two ternary signals as follows: 0 = [00], 1 = [01], 2 = [02] and 3 = [1X], where X means any of values 10, or 11, or 12. Thus, our encoding method introduces the don't cares and in general the relations to the MV data for decomposition. It proves thus that the concept of decomposing relations, introduced by us for Machine Learning and circuit design applications in program GUD-MV [12,13], is also useful for fuzzy logic. Currently we keep

looking for more fuzzy logic benchmarks, especially large ones, but unfortunately all examples from books and conference proceedings that we were able to find are too small for the power of our decomposers. Perhaps the answer to this problem is to create large fuzzy data on our own. We intend to generate them automatically as the results of image processing procedures that create fuzzy features for pattern recognition experiments. Next our Constructive Induction approach to Machine Learning based on uniform approach to the decomposition of binary multi valued and fuzzy functions will be used in the final stage of pattern recognition instead of a Gaussian Classifier that we currently use [15,16]. Currently we are able to generate automatically multi-valued functions and relations from robot data (image and sensors) [24].

9. **CONCLUSION**

The new method of converting fuzzy functions to multiple-valued functions for decomposition allows not only for Ashenhurst-like, but also for Curtis-like decompositions. By converting fuzzy functions to multiple-valued functions we eliminate the time-consuming conversion to the canonical form. The need for special and complex methods like Kandel's decomposition method does no longer exist, and any existing MV decomposer can be used. Thus, various decomposers lead to different kinds of fuzzy functions decompositions. Our method can be expanded to arbitrary shape of fuzzy literals, and not only the literals x discussed above. Such an extension leads to multi-valued encodings of these fuzzy functions with logic radices higher than 3. In addition, our method can be used with no modification to relations. Several decomposers [12 -- 24] can be used for this task, again leading to different decompositions that can be evaluated and compared.

REFERENCES

[1] D. R. Luginbuhl and A. Kandel (1983) "A comparison of fuzzy switching functions and multiple-valued switching functions," Proc. IEEE International Symposium of Multiple-Valued Logic, 264 – 272.

[2] X. Zhiwei (1984) "Multivalued logic and fuzzy logic - their relationship, minimization, and application to fault diagnosis," IEEE Transactions on Computers, C-33(7), 679 - 681.

[3] A. Kandel and J. M. Francioni (1983) "Decomposable fuzzy valued switching functions," Fuzzy Sets and Systems, 9, 41- 68.

[4] L. A. Zadeh (1972) "A Fuzzy Set Theoretic Interpretation of Linguistic Hedges," J. Cybern., 2, 3, 4–34.

[5] M. Sakawa (1993) "Fuzzy Sets and Interactive Multiobjective Optimization," Plenum Press, NY,NY.

[6] A. Kandel (1974) "On the Minimization of Uncompletely Specified Fuzzy Functions," Information and Control, 26, 141-153.

[7] P. N. Marinos (1969), "Fuzzy logic and its application to switching systems," IEEE Transactions on Computers, C-18(4), 343-348.

[8] M. Karnaugh (1953) "The map method for Synthesis of Combinational Logic Circuits," Transactions AIEEE, 72, pt. I, 593 – 598.

[9] G. W. Schwede, and A. Kandel (1977) "Fuzzy maps," IEEE Trans. SMC-7(9), 699 – 674.

[10] E. W. Veitch, (1952) "A Chart Method for Simplifying Truth Functions," Proc. ACM, 127 – 133.

[11] M. Mukaidono (1984) "An Improved Method for Minimizing Fuzzy Switching Functions," Proc. IEEE International Symposium of Multiple-Valued Logic, 196 – 201.

[12] S. Grygiel and M. Perkowski (2001) ``Labeled rough partitions - a new general purpose representation for multiple-valued functions and relations", Journal of Systems Architecture, Vol. 47, 29-59.

[13] M. Perkowski, M. Marek-Sadowska, L. Jozwiak, T. Luba, S. Grygiel, M. Nowicka, R. Malvi, Z. Wang, and J. Zhang (1997) "Decomposition of Multi-Valued Functions and Relations," Proc. IEEE International Symposium of Multiple-Valued Logic.

[14] C. Files, R. Drechsler, and M. Perkowski (1997) "Functional Decomposition of MVL Functions using Multi-Valued Decision Diagrams," Proc. IEEE Intern. Symp. Multiple-Valued Logic,

[15] M. A. Perkowski, S. Wang, W. K. Spiller, A. Legate, and E. Pierzchala (1990) "Ovulo-Computer: Application of Image Processing and Recognition to Mucus Ferning Patterns," Proc. of the Third IEEE Symposium on Computer-Based Medical Systems, 52-59, Chapel Hill, North Carolina.

[16] M. A. Perkowski, S. Wang, W. K. Spiller, A. Legate, and E. Pierzchala (1990) "A Simple and Portable PC-based Image Processing System and its Application in Ovulometry," Proc. Imaging West Conference, 758-763, Pasadena, CA.

[17] A. Mishchenko, B. Steinbach, and M. Perkowski (2001) ``An Algorithm for Bi-Decomposition of Logic Functions," Proceedings of Design Automation Conference, DAC 2001, June 18-22, Las Vegas, 103 - 108.

[18] A. Mishchenko, B. Steinbach, and M. Perkowski (2001)``Bi-Decomposition of Multi-Valued Relations," Proc. 10-th International Workshop on Logic and Synthesis, IWLS'01 , 35 – 40.
[19] M. Perkowski, and S. Grygiel (2001) `` Decomposition of Relations: A New Approach to Constructive Induction in Machine Learning and Data Mining - An Overview" Proc. Workshop National Institute of Telecommunications and Polish Association for Logic and Philosophy of Science, 91 - 111, Warsaw, Poland.
[20] M. Perkowski, R. Malvi, S. Grygiel, M. Burns, and A. Mishchenko (1999) ``Graph Coloring Algorithms for Fast Evaluation of Curtis Decompositions," *Proc. DAC'99.*
[21] B. Steinbach, M. Perkowski, and Ch. Lang (1999) ``Bi-Decomposition of Multi-Valued Functions for Circuit Design and Data Mining Applications," Proc. ISMVL'99*,* 50 – 58.
[22] C. Files, and M. Perkowski (1998) ``Multi-Valued Functional Decomposition as a Machine Learning Method," Proc. ISMVL'98.
[23] C. Files, and M. Perkowski (1998) ``An Error Reducing Approach to Machine Learning Using Multi-Valued Functional Decomposition," Proc. ISMVL'98.
[24] U. Wong and M. Perkowski (2002) "A New Approach to Robot's Imitation of Behaviors by Decomposition of Mutiple-Valued Relations," Proc. 5th International Workshop on Boolean Problems, Freiberg University of Mining and Technology, Freiberg, Germany, 265-279.

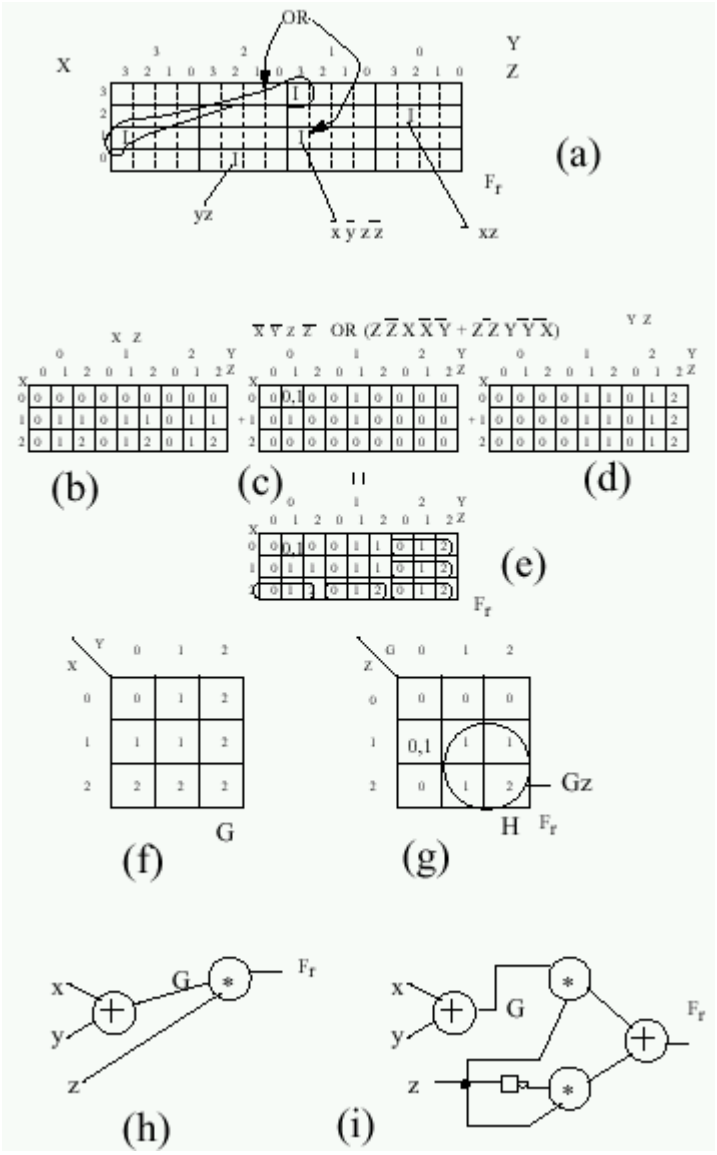


Figure 15. Stages of decomposition of a fuzzy relation to Example 8. (a) Original fuzzy relation F_r. (b) – (e) stages of creating a ternary relation corresponding to fuzzy relation F_r, (f) ternary function G from decomposition, (g) ternary function H from decomposition, (h) (i) Two realizations of fuzzy relation F_r , corresponding to two realizations of ternary relation H.