# Fuzzy weakest precondition and algorithm specification

**Victoria López**[1], **Daniel Gómez**[2], **Javier Montero**[1]
[1]Dept. Estadística e I.O. I. Complutense University. Email {vlopez,javier_montero}@mat.ucm.es
[2]Dept. Estadística e I.O. III. . Complutense University. Email dagomez@estad.ucm.es

## Abstract

Design and formal verification of algorithms can be translated into a fuzzy framework introducing fuzzy logic and assert transformations. Following the classical scheme, and in order to develop codes with good behavior, this paper defines the concepts of fuzzy algorithm specification and fuzzy weakest precondition operator which are then applied to a fuzzy decision making algorithm.

**Keywords:** Fuzzy algorithms and programming, decision making.

## 1 Introduction

An algorithm is a precise, step-by-step method of doing a task in a finite amount of time. In general, an algorithm should be previously checked in order to assure that such algorithm has been rightly designed. This has to be done by means of a formal specification [5], which refers to two different aspects:

- Precondition describing the situation before algorithm execution (any assumption and the information the algorithm needs to meet its objectives).
- Postcondition describing the situation after algorithm execution (any error condition and the information returned by the algorithm).

Both pre and postcondition, are logic formulas. In this paper we shall assume that precondition can include fuzzy restrictions. A first study of the design of the associated *fuzzy algorithm* has been developed in [9], including the possibility of a fuzzy postcondition (in particular, it was developed an iterative code for the algorithm, that can be verified by means of mathematical induction and transformations of logic assertions, which set the relationship between variables and values involved in the program).

Verification processes should play a more important role within the fuzzy context than within the crisp context, since it may be the only available checking tool for some decision making problems, when the concepts of *success* and *fail* can not be properly defined.

## 2 Algorithm specification

An algorithm can be formally described by its formal specification, that means three parts: header (a pseudocode program header associated), precondition and postcondition. All first logic formulas will be between slice brackets in order to separate then from sentences of code.

**Definition 2.1** *An algorithm is called 'fuzzy' when its specification admits fuzzy conditions.*

An example of a crisp algorithm specification is now given, followed by a fuzzy one.

**Example 2.1** *An array of n integers is called 'Blond' if at least one of its component is odd. An*

*algorithm for deciding if a given array is 'Blond' can be described by the following (crisp) specification:*

**fun** $Blond$ *( a: vector )* **ret** *b:* ***boolean****;*
{*PRE:* $n \geq 0$}
{*POST:* $b = \exists i \in \{1..n\} : a[i] mod 2 = 1$}
*where it is used the type*

*vector=**array**[1..n] **of integer***

**Example 2.2** *An array of integers is 'Blond' if it has at least one blond component. An integer is 'Blond' if is 'near to 4'. The aim of the following algorithm is to know how blond is the given array. The algorithm can be described by its fuzzy specification:*

**fun** $fBlond$ *( a: vector;* $\mu_4$*: 01-function )*
**ret** *b: [0,1];*
{*fPRE:* $n \geq 0$}
{*fPOST:* $b = \max_{i \in \{1..n\}} \mu_4(a[i])$}
*where* $\mu_4$ *given as an input is the fuzzy concept of 'near to 4'.*

## 3 Top-down and bottom-up formal correctness

An iterative algorithm consists of a sequence of k instructions

$$I_1; I_2; \ldots I_k;$$

Formal verification process must consider assertions between each couple of instructions in order to determine local pre and postconditions that describe the configuration before and after instructions:

$$\{\varphi_{i-1}\} I_i; \{\varphi_i\}$$

*Top-down formal correctness* considers the precondition as the initial configuration, then compute next configuration as the result of the execution of the following instruction of code until the last one:

$$\{PRE\} I_1; \{\varphi_1\} I_2; \{\varphi_2\} \ldots I_k \{\varphi_k\} \Rightarrow \{POST\}$$

After that, the last configuration $\{\varphi_k\}$ must imply the algorithm postcondition ({POST}) as it is shown above.

*Bottom-up formal correctness* works in the opposite way:

$$\{PRE\} \Rightarrow \{\varphi_0\} I_1; \{\varphi_1\} I_2; \{\varphi_2\} \ldots I_k \{POST\}$$

Bottom-up formal correctness starts in postcondition and computes the precondition of $I_k$, denoted by $\{\varphi_{k-1}\}$ and so on, until assertion $\{\varphi_0\}$ is reached. After that, the algorithm precondition ({PRE}) must imply $\{\varphi_0\}$.

Bottom-up formal correctness is the usual approach, since Dijkstra [3] proved that there exists a suitable operator for transforming logic assertions. This operator is introduced in the following section.

## 4 The fuzzy weakest precondition operator

The importance of formal specification and verification in algorithm design has been already pointed out by Dijkstra [3], who also introduced the weakest precondition operator [4]. The first idea consists of assigning a triple

$$(\varphi, I, \phi)$$

to each instruction of code $I$ in a standard programming language. Both $\varphi$ and $\phi$ are assertions that describe the situation before and after executing the instruction I. Then $\varphi$ is the precondition and $\phi$ is the postcondition of the single $I$. The local precondition $\varphi$ can be computed from instruction $I$ and its local postcondition $\phi$.

**Definition 4.1** *Weakest precondition operator is a function*

$$wp : \textbf{\textit{Assertions}} \times \textbf{\textit{Statements}} \rightarrow \textbf{\textit{Assertions}}$$

*where* ***Assertions*** *is the set of first order logic formulas;* ***Statements*** *is the set of instructions in a standard programming language, usually denoted by* $I$*. Weakest precondition allows to obtain the first order formula*

$$\varphi = wp(I, \phi)$$

*such that the sequence of code*

$$\{\varphi\}\ I\ \{\phi\}$$

*is formally correct. That means if the configuration holds the precondition then the postcondition* $\phi$ *will be reached after executing instruction* $I$*, i.e. if statement* $I$ *is executed when configuration* $\varphi$ *is held, then the configuration became* $\phi$*, that can also be represented as*

$$\varphi \circ I = \phi$$

*If any other formula $\varphi'$ is such that*

$$\{\varphi'\}\ I\ \{\phi\}$$

*is also formally correct then*

$$\varphi' \Rightarrow \varphi$$

*In this sense, $\varphi$ is the weakest.*

**Example 4.1** *Let us consider the following sequence:*

$\{x > 0\}$
**if** $x \geq 0$ **then** $s := 1$
**else** $s := -1$
**endif;**
$\{x > 0 \wedge s = 1\}$

*Precondition, if-instruction and postcondition here are directly related. Nevertheless pre and postcondition are not always so obvious.*

It is normal to get the weakest precondition by means of the inference rule [9] related to its own statement $I$. The inference rule for assignment instruction establishes that

$$wp(x := Exp, \{\phi\}) = \{\phi_x^{Exp}\}$$

being $\{\phi_x^{Exp}\}$ the first order formula $\{\phi\}$ after replacing any instance of variable $x$ with expression $Exp$.

**Example 4.2** *Let us consider the following sequence:*

$$\{\varphi\}\ x := x + 1;\ \{\phi \equiv (x = 2y + 1)\}$$

*the weakest precondition is:*

$$\{\varphi\}\ \equiv \{\phi_x^{x+1}\} \equiv \{x + 1 = 2y + 1\} \equiv \{x = 2y\}$$

**Definition 4.2** *Let $I$ be a simple instruction of code and let $\phi$ be a fuzzy first order logic formula. The* fuzzy weakest precondition

$$fwp(I, \phi) = \varphi$$

*for obtaining $\phi$ after executing $I$*

$$fwp : \textbf{\textit{fAssertions}} \times \textbf{\textit{Statements}} \rightarrow \textbf{\textit{fAssertions}}$$

*is another fuzzy assertion $\varphi$ obtained as the result of applying the inference rule for $I$, in such a way that $\varphi$ is the weakest condition needed to satisfy $\phi$ after running $I$.*

***fAssetions** refers to the set of fuzzy first order logic formulas.*

As a special case, the fuzzy weakest precondition

$$fwp(x := Exp, \phi)$$

can be obtained by replacing in $\phi$ every item $x$ by the expression $Exp$. Then, the sequence

$$\{fwp(x := Exp, \phi)\}\ x := Exp;\ \{\phi\}$$

is formally correct.

After that, the new assertion $\varphi$ must be simplified using the fuzzy operators suitable for the specifical algorithm. Some T-norm $T$ and a T-conorm $S$ must be selected at the beginning, according to the objective and task [9].

**Example 4.3** *Given two groups of people $X$ and $Y$, let $\varphi_1, \varphi_2, \varphi_3$ be the following fuzzy logic formulas:*

- *$\varphi_1(X)$ $\equiv$ 'Most members in $X$ are born in Spain'*
- *$\varphi_2(Y)$ $\equiv$ 'Most members in $Y$ are born in France'*
- *$\varphi_3(X)$ $\equiv$ 'About a half of members in $X$ are born in Spain'*

*Let us consider the following code:*

$\{\varphi\}$
$X := X \cup Y;$
$\{\phi\} \equiv \{\varphi_2(Y) \wedge_f \varphi_3(X) \wedge_f |X| = 2|Y|\}$

*where $\wedge_f$ is a fuzzy operator for $\wedge$ ($Min$, for instance). The fuzzy weakest precondition solves the equation by substitution:*

$$\varphi = fwp(X := X \cup Y; \phi)$$

$$= \varphi_2(Y) \wedge_f \varphi_3(X \cup Y) \wedge_f |X \cup Y| = |Y|$$

$$= \varphi_2(Y) \wedge_f \varphi_1(X) \wedge_f |X| = |Y|$$

*And the solution is a new fuzzy logic assertion, according to the set of assumptions relative to fuzzy operators and fuzzy relations values for $\varphi_1, \varphi_2, \varphi_3$. So, fuzzy weakest precondition will be the aggregation of the above assertions, that can be interpreted as 'X and Y have about the same number of members, most members in X are born in Spain and most members in Y are born in France'.*

## 5 Application to a fuzzy decision making algorithm

Classification processes [1, 2] are usually implemented by means of algorithms based on the process itself.

An algorithm which solves a classification problem is usually evaluated according to the associated classification process. If the process is good, we consider that the algorithm associated is also good. Nevertheless, good behavior of the algorithm is only guaranteed by a formal verification of the code line by line. This is the only way of assuring that postcondition is going to be reached from a precondition, i.e., the formal specification of the process.

Our problem is how to decide if a classification of the set of pixels in a digital image, for instance, is good or not (crisp case), or if it is good enough or not (fuzzy case) [7].

In order to design a good algorithm that make a decision about a classification [8], a formal specification is due. A non-formal specification can include the header, precondition and postcondition expressed in natural language:

**proc** $C\&Decide$ (**in** $im$: Timage;
**out** $imc$:Tlist **of** Timage; **out** $k$: **int**);

{fPRE: $im$ has all the properties of a image }

{fPOST: $imc$ is a list of matrix that represents
the result of a good or good
enough classification
of the set of pixels in $im$.
Variable $k$ returns the length of $imc$}

The next table shows the formal header and the pseudo-code we propose in order to solve the problem.

**proc** $C\&Decide$ (**in** $im$: Timage;
**in** $\alpha_{int}$ : $[0, 1]$;
**out** $imc$:Tlista **of** Timage;
**out** $k, k_0$: **int**;
**out** $\alpha_{res}$: [0,1]);
$\{$fPRE $_{C\&Decide}\}$
*CrispC(im,imc,k)*;
$k_0 := k$;
*Decide(im, imc, k, nk, ok,*$\alpha_{res}$*)*;
**while** $\neg ok$ **do**
$FuzzyC(im, nk, imc, k)$;
$Decide(im, imc, k, \alpha_{int}, nk, ok, \alpha_{res})$;
**endwhile**;
{fPOST $C\&Decide$}
**endproc**;

Firstly, the algorithm develops a crisp classification of the set of pixels by means of an internal process called $CrispC$. After that, the algorithm calls to $Decide$ in order to know if our expert (the '*internal expert*') considers 'good enough' the last classification. If her/his answer, $ok$, is not 'True', a loop starts running two subprocess: a call to $FuzzyC$ for getting a new fuzzy classification and a new call to $Decide$ again, to know the expert opinion about. The loop iterates until getting her/his approval.

Precondition in this procedure demands properties about the digital image. In this paper we assume the image is a RGB one, and it is represented by a matrix of pixels (see [1, 2]).

$\{fPRE_{C\&Decide}$ :
$Image(im) \equiv$
$\forall (i, j) \in \{1..n\} \times \{1..m\} : im[i, j] \in \{0..255\}^3\}$
$\wedge (0 \leq \alpha_{int} \leq 1)\}$

Notice that, in this case, precondition is a crisp one.

Postcondition gives a definition of $imc$ as a fuzzy classification of the input. In this sense, we can make the following first order fuzzy formula:

$$\{fPOST_{C\&Decide}:$$
$$[CrispC(imc, im) \vee_f FuzzyC(imc, im)]$$
$$\wedge_f (Value_{int}(imc, im) \succeq \alpha_{int})$$
$$\wedge_f (k = lenth(imc))\}$$

Since fuzzy logic generalizes the traditional logic, this formula will be valid or true enough when the loop is not going to be done (crisp classification) and when the loop is going to be done (fuzzy classification).

The clause, $k = lenth(imc)$, is a crisp one so that we avoid a fuzzy final number of classes. However, the clause $(Value_{int}(imc, im) \succeq \alpha_{int})$ refers to a degree of validity of the final classification $imc$ for the image $im$, which is compared to $\alpha_{int}$ that represents a goodness degree given as input. Since this comparation can be a preference relation, the result will be a real value between 0 and 1.

The loop stops depending on the internal expert decision, who must agree after a reasonable number of iterations. Moreover, we can also develop an alternative algorithm including some bound for the number of iterations, in such a way is assured the end of the loop.

All of these considerations must conform an exhaustive analysis of the design without forgetting the aim of correctness. Each family of fuzzy operators $\wedge_f$ and $\vee_f$ force a particular fuzzy semantics, and results will be extremely dependent on this election. Therefore, choosing them is a key issue, and some learning procedure will be needed in order to clarify specification true values and other desired properties, to be taken into account in each specific problem.

## 6 Final remarks

Algorithm correctness or algorithm verification appears as an absolute need when specifications contains fuzzy relations in any software project, where a good design will improve programs and regular output testing does not represent a guarantee for the right behavior of the program. This is specially the case when a fuzzy precondition appears. Implementation of fuzzy algorithms requires much more effort in developing verification techniques, some times the only reliable tool in order to check decision processes in a fuzzy framework.

## Referencias


[1] A. Amo, J. Montero, A. Fernández, M. López, J. Tordesillas and G. Biging, "Spectral fuzzy classification: an application". *IEEE Trans.Syst.Man.Cyb.* (C) 32, 42-48, 2002

[2] A.Amo, D. Gómez, J. Montero, "Spectral fuzzy classification: a supervised approach". *Mathware and Soft Computing* 10:141-154, 2003

[3] E.W. Dijkstra, "A discipline of Programming". *Prentice Hall*, 1976.

[4] E.W. Dijkstra, W.H.J. Feijen, "A method of programming". *Addison-Wesley*, 1988.

[5] H.K. Berg, "Formal Methods of Program Verification and Specification". *Prentice Hall*, New Jersey, 1982.

[6] M. Broy y B. Krieg, "Derivation Of Invariant Assertions During Program Development by Transformation". *ACM Transactions on Programming Languages and Systems*, Vol. 2, N 3, 321–327 1980.

[7] V. López y L. Martín, "Coloreado de grafos difusos: Verificación formal y comprobación prctica con MAPLE", *Proceedings ESTYLF conference* Jaen, Spain, September 15-17, 2004, pages 545-550.

[8] V. López, J. Montero y D. Gómez, "Verificación de algoritmos de clasificación con precondición difusa", *Proceedings ESTYLF conference* Jaen, Spain, September 15-17, 2004, pages 437-442.

[9] V. López, "Diseño y verificación de algoritmos para el tratamiento difuso de imágenes digitales en teledetección y seguridad". *Ph.D. Thesis Politechnic University of Madrid*, Spain, 2004 (in Spanish).