On a class of fuzzy computable functions

Rafael Morales-Bueno *, Ricardo Conejo, José-Luis Pérez-de-la-Cruz, Francisco Triguero-Ruiz

Dpto. Lenguajes y Ciencias de la Computación, ETSI Informática, Universidad de Málaga, Bulevar Luis Pasteur s/n, Málaga 29071, Spain

Received 22 February 1999; received in revised form 15 October 1999; accepted 15 October 1999

Abstract

We present in this paper a finitary approach to the concept of fuzzy computability. On this basis we define the class $FR$ of fin-recursive $W$-functions. We prove that $FR$ is a strict subset of the class of $W$-functions with recursive graph, as defined by Gerla. We show that it is possible to generate the class $FR$ by means of programs written in the language $XL$. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Fuzzy relations; Fuzzy programming; Fuzzy computability; Fuzzy languages

1. Introduction

Fuzzy computability is a somewhat complex subject. There have been many proposals to define computability [3,4,6,15], and some of them are too broad. A first clarification was made by Gerla [5] with his hierarchy of classes. However, from a certain point of view, even Gerla’s smallest class $WRG$ might be considered too extensive. In fact, since the complete image of an element $u$ can be infinite, a finite machine cannot compute it in finite time. For this reason, in this paper a new class $FR$ of $W$-functions that refines Gerla’s hierarchy ($FR \subset WRG$) is defined. This class will be called the class of finitely recursive $W$-functions or, in short, fin-recursive functions. The conditions for a function with recursive graph to be fin-recursive are also studied in this paper.

In recent papers [10,12], we have introduced the fuzzy programming language $XL$ with a small set of instructions; one of them is the fuzzy assignment. As it was pointed out there, it makes sense to study the possibilities arising when only a restricted finite subset of $W$-computable functions is allowed in the assignment, both from a practical and a theoretical point of view, in order to generate a given class of functions. When a $W$-function $f$ can be computed by an $XL$-program whose fuzzy assignments make use of...
a set $F$ of $W$-functions, we say that $f$ is $XL(F)$-programmable. We will define a certain finite set $C$ and prove that a $W$-function $f \in FR$ iff $f$ is $XL(C)$-programmable.

The rest of this paper is structured as follows: in Section 2 some definitions concerning fuzzy Turing machines are presented and the class $FR$ is defined in terms of these concepts. Some properties of $FR$ are also proven. In Section 3 a short description of the $XL$ language is given for the sake of completeness of this paper and the relationships between $XL$ programs and multitape fuzzy Turing machines are shown. In Section 4 the class $FR$ is defined in another way, via $XL$-programs. Finally, in Section 5 some conclusions are drawn and future lines of research are suggested.

2. Fuzzy Turing machines and finitely recursive functions

In the following, we will consider that the degrees are evaluated in a finite ordered semiring $W(\{l_0, \ldots, l_r\}$, $\ominus, \otimes, \leq$). We will assume that the identity elements are $l_0$ for $\oplus$ and $l_1$ for $\otimes$. Sometimes we will write 1 for $l_1$ and 0 for $l_0$.

Let $\mathbb{Z}^+$ be the set of nonnegative integers. We will use $W$-sets over $(\mathbb{Z}^+)^n$, $n \geq 1$, with the standard representation: $A = \mu_a(a)/a + \cdots + \mu_a(z)/z$. If $a = (a_1, \ldots, a_n) \in (\mathbb{Z}^+)^n$ then $\pi_i$ represents the $i$th-projection $\pi_i(a) = a_i$.

Let $U$ and $V$ be sets. A $W$-function $f$ from $U$ into $V$ is a function from $U \times V$ into $W$. If $f(u, v) = \mu$, then we say that the value of the $W$-function $f$ at $u$ is $v$ with weight or degree $\mu$. The image $Im(f, u)$ of $f$ at $u$ is a subset of $V$ given by $Im(f, u) = \{v : f(u, v) \neq 0\}$.

For the concept of $W$-Turing machine (in the following, $W$-$TM$), we will use the definition of [1] with some minor modifications.

**Definition 1.** Let $W$ be an ordered semiring. A $W$-$TM$ is a 4-uple $Z = (V, S, p, s_0)$ where $V$ and $S$ are finite nonempty sets, $V \cap S = \emptyset$; $s_0 \in S$ and $p$ is a $W$-function from $S \times V$ to $(V \cup \{L, R\}) \times (S \cup \{h\})$; $L, R \notin V$, $h \notin S$.

In the definition above $S$ is the state set, $s_0$ the initial state, $V$ the set of symbols. Moreover, $p$ is the transition $W$-function, namely $p(s, C, z, s')$ which represents the degree of validity of the choice of the act $(z, s')$ at state $s$ and when the tape symbol $C$ is scanned. Such an act is to move to the right (if $z$ is $R$), to the left (if $z$ is $L$) or to replace $C$ by $z$ (if $z \notin V$) and successively to assume the state $s'$ (stop if $s'$ is $h$). We shall assume that in $V$ there is a special symbol $\#$ that stands for blank.

For every $u \neq 0$, we can write $(s, C) \xrightarrow{\mu}(z, s')$ to denote that $p(s, C, z, s') = u$ and $(s, C) \rightarrow (z, s')$ to denote that $p(s, C, z, s') = 1$. The $W$-function $p$ is characterized by a finite set of arrows of this type.

**Definition 2.** Let us define $V = \{a : a \in V\}$. The empty string will be denoted by $\epsilon$. An elementary description is an element of $(S \cup \{h\}) \times (\{\epsilon\} \cup (V - \{\#\}) \cup (V^* (V - \{\#\}) \cup \{\epsilon\}))$, i.e., a pair of the form $(s, uaw)$, where $s$ is a state (or the halting state $h$), $u$ is the empty string or a string over $V$ whose first symbol is not $\#$, $a$ is a symbol of $V$, and $w$ is the empty string or a string over $V$ whose last symbol is not $\#$. An elementary description expresses the present state ($s$), the content of the tape ($uaw$) and the position and content of the scanned square ($a$).

Elementary descriptions will be denoted by $d, d_1, d_2, \ldots$.

**Definition 3.** An elementary description $(s, uaw)$ is final iff $s = h$.
Let $D(T)$ be the set of elementary descriptions for the $W$-$TM$ $T=(V,S,p,s_0)$. We will write $D$ for $D(T)$ when $T$ can be clearly determined from the context.

Now, the $W$-function $p^*$ can be defined. This function describes one step of computation.

**Definition 4.** Let $T=(V,S,p,s_0)$ be a $W$-$TM$. Let $D$ be the set of elementary descriptions of $T$. The $W$-function $p^*:D \rightarrow D$ is defined as follows: for each $d,d' \in D$

$$p^*(d,d') = \begin{cases} 
(p(s,c,c',s')) & \text{if } d = (s,ucw) \text{ and } d' = (s',uc'w)'w, \\
(p(s,c,R,s')) & \text{if } d = (s,ucw) \text{ and } d' = (s',ucc'w) \text{ or } d = (s,uc) \text{ and } d' = (s',uc'h), \\
(p(s,c,L,s')) & \text{if } d = (s,uc'cw) \text{ and } d' = (s',uc'cw) \text{ or } d = (s,cw) \text{ and } d' = (s',#cw), \\
0 & \text{otherwise},
\end{cases}$$

where # is the blank, $u,w \in V^*$, $s,s' \in S$ and $C,C' \in V$.

**Comment.** Santos [15] defines a computation $\mu$ with input $x \in V^*$ and output $y \in V^*$ as a finite sequence of elementary descriptions $\mu = d_1d_2\ldots d_n$ where $d_1 = (s_0,x\#)$, $y$ is obtained from $d_n$ by erasing all symbols not in $V^*$, and $d_1$ is a final elementary description. The weight associated with a computation $\mu = d_1d_2\ldots d_n$ is defined by

$$w(\mu) = p^*(d_1,d_2) \otimes p^*(d_2,d_3) \otimes \cdots \otimes p^*(d_{n-1},d_n).$$

Following this, he denotes by $w(T,x,y)$ the multiset (possibly infinite) of the weights of all the computations with input $x$ and output $y$, and defines $\bigoplus w(T,x,y)$ as the least upper bound of the partial sums of the elements in $w(T,x,y)$. Then, a $W$-function $f$ is $T$-computable if a $W$-$TM$ exists such that $f(x,y) = \bigoplus w(T,x,y)$. Notice that, since $w(T,x,y)$ may arise from computations of unbounded length, no configuration computed in any number of steps is guaranteed to have a degree equal to the corresponding degree in $\bigoplus w(T,x,y)$. That is to say, no computed degree $\alpha$ can be assumed to be the actual degree of $f(x,y)$.

Our proposal here is to add the condition of total termination, i.e., to request that – sooner or later – all elementary descriptions in a description become final elementary descriptions. The $W$-functions computable by a $W$-$TM$ satisfying this additional request will be called fin-recursive $W$-functions.

The preceding informal remarks can be easily formalized. Let $T=(V,S,p,s_0)$ be a $W$-$TM$ and $D(T)$ the set of its elementary descriptions. We will denote the fuzzy subsets of $D(T)$ by $\Delta(T)$, $\Delta_1(T)$, $\Delta_2(T)$, $\ldots$. Each $\Delta(T)$ will be called a description. The machine $T$ can be omitted if it is clear from the context.

**Definition 5.** $\Delta$ is an initial description iff $\Delta = 1/(s_0,x\#)$. $\Delta$ is a final description iff

1. $\mu_\Delta(d) \neq 0$, for at least one $d \in D(T)$,
2. For every $d \in D(T)$, with $\mu_\Delta(d) \neq 0$, $d$ is a final elementary description.

We can now define the function that computes a description $\Delta'$ in one step. Since only a finite number of elementary descriptions can be generated in a finite number of steps, the definition assumes that the support of $\Delta$ is finite.

**Definition 6.** Let $T$ be a $W$-$TM$ and let $\Delta = x/d$ and $\Delta' = x_1/d_1 + \cdots + x_s/d_s$ be descriptions of $T$. We say that $T$ computes $\Delta'$ from $\Delta$ in one step, and write $\Delta \Rightarrow_T \Delta'$, iff:

(a) $x \otimes p^*(d,d_i) = x_i$, $i = 1,\ldots,s$,
(b) $\forall d' \neq d_i$, $i = 1,\ldots,s$, $p^*(d,d') = 0$,
(c) if $d$ is a final elementary description, then $\Delta = \Delta'$. 

Definition 7. Let $T$ be a $W$-TM and let $A = a_1/d_1 + \cdots + a_r/d_r$ and $A'$ be descriptions of $T$. We say that $T$ computes $A'$ from $A$ in one step, and write $A \Rightarrow_T A'$, iff

(a) for each $i = 1, \ldots, r$, $a_i/d_i \Rightarrow_T A_i$, 
(b) $A' = \bigcup_{i \in \{1, \ldots, r\}} A_i$, where the union is defined w.r.t. the operation $\oplus$.

We define $\Rightarrow_T^*$ as the reflexive transitive closure of $\Rightarrow_T$.

Comment. Definition 6 states the procedure to compute descriptions from elementary descriptions by means of the $W$-function $p^*$. Definition 7 states the procedure to compute descriptions from descriptions as the union of the descriptions generated from every elementary description. The $\oplus$ operator is used to integrate the degrees.

Definition 8. Let $T$ be a $W$-TM. $T$ halts with input $x$ iff there exists a final description $A$ such that $1/(s_0, x^\#) \Rightarrow_T^* A$.

Definition 9. Let $I$ and $O$ be alphabets. Let $f$ be a $W$-function from $I^*$ to $O^*$. We say that the $W$-TM $T = (V, S, p, s_0)$ constructs $f$ iff

(a) $I, O \subseteq V$. 
(b) For each $x \in I^*$, $T$ halts with input $x$ and the final description is $A$ such that $f(x, y) = \mu_A(d)$, with $d = (h, y^\#)$.

A $W$-function $f$ is fin-recursive iff there exists a $W$-TM that constructs $f$. $FR(I^*, O^*)$ will denote the class of all fin-recursive $W$-functions from $I^*$ to $O^*$.

Comment. Informally speaking, we are describing the following process: A $W$-TM starts from the initial state with the input $x$ written in the tape. Due to the fuzzy nature of the machine, many computations are carried out from this, beginning with different degrees. However, after a finite number of steps every computation halts and the different contents of the tape are exactly the $y$’s such that $f(x, y) \neq 0$. Moreover, the degrees of the halting computations are exactly $f(x, y)$.

Note that in this way we could define a crisp construct by setting $W = \{0, 1\}$. Then the $W$-TM would be in fact a nondeterministic TM and the class of fin-recursive functions would be the class of crisp multivaluated functions computed by a nondeterministic TM such that it halts along every path of computation. This process is essentially different from Santos’ construct. In fact, Santos does not impose the halting condition; then, nonterminating computations can exist, and an external nonoracular observer cannot decide if a certain $y$ will ever appear in the computations, so yielding a value $f(x, y) \neq 0$. He cannot decide even if the degree of a certain $f(x, y)$ – computed in a finite number of steps – will be modified.

Now, we define the class of fin-recursive $W$-functions from $\mathbb{Z}^+$ to $\mathbb{Z}^+$.

Let $|$ be some fixed symbol other than the blank symbol; then the positive integer number $n$ may be represented by the string $|^n$. (Thus zero is represented by the empty string.)

Definition 10. A $W$-function $f$ from $\mathbb{Z}^+$ to $\mathbb{Z}^+$ is said to be constructed by a $W$-TM $Z$, if $Z = (V, S, p, s_0)$, $V = \{|, \#\}$, and $Z$ constructs the $W$-function $f'$ from $V^*$ to $V^*$, where $f'(|^n) = |^{f(n)}$ for each $n \in \mathbb{Z}^+$. $FR(1, 1)$ will denote the class of all fin-recursive $W$-functions from $\mathbb{Z}^+$ to $\mathbb{Z}^+$. 
Example 1. Let $W$ be $\{l_0, l_2, l_3, l_4, l_1\}$ (remember that $l_0$ is the identity for $\oplus$ and $l_1$ is the identity for $\otimes$).

Let us show a $W$-TM that constructs the following $W$-function $g$ from $\mathbb{Z}^+$ to $\mathbb{Z}^+$: for every $n \in \mathbb{Z}^+$

\[

g(n, n) = l_2, \\
g(n, n + 1) = l_1, \\
g(n, n + 2) = l_4, \\
g(n, n + 3) = l_3,
\]

We define $Z = (V, S, p, s_0)$, where $V = \{|, \#\}$, $S = \{s_0, s_1, s_2\}$ and $p$ is given by

1. $p(s_0, \#, \#, h) = l_2$,
2. $p(s_0, \#, |, s_0) = l_1$,
3. $p(s_0, |, R, h) = l_1$,
4. $p(s_0, |, R, s_1) = l_1$,
5. $p(s_1, \#, |, s_1) = l_1$,
6. $p(s_1, |, R, h) = l_4$,
7. $p(s_1, |, R, s_2) = l_1$,
8. $p(s_2, \#, |, s_2) = l_1$,
9. $p(s_2, |, R, h) = l_3$.

Let us assume that the input is the number 3, i.e., the initial description is

\[A_0 = l_1/(s_0, ||\#).\]

Transitions (1) and (2) yield a new description with two elementary descriptions:

\[A_1 = l_1/(s_0, ||\#) + l_2/(h, ||\#).\]

The following chain of descriptions represents the computation of the machine:

\[
A_2 = l_1/(s_1, ||\#) + l_1/(h, ||\#) + l_2/(h, ||\#), \\
A_3 = l_1/(s_1, ||\#) + l_1/(h, ||\#) + l_2/(h, ||\#), \\
A_4 = l_1/(s_2, ||\#) + l_4/(h, ||\#) + l_1/(h, ||\#) + l_2/(h, ||\#), \\
A_5 = l_1/(s_2, ||\#) + l_4/(h, ||\#) + l_1/(h, ||\#) + l_2/(h, ||\#), \\
A_6 = l_3/(h, ||\#) + l_4/(h, ||\#) + l_1/(h, ||\#) + l_2/(h, ||\#).
\]

In this way, we compute $g$ for the input 3, i.e., $l_3/6 + l_4/5 + l_1/4 + l_2/3$.

We prove now some properties of the class $FR(I^*, O^*)$.

**Proposition 1.** Let $f$ be a fin-recursive $W$-function from $I^*$ to $O^*$. Then for every $x \in I^*$, $Im(f, x)$ is a finite non-empty set.

**Proof.** Let $T = (V, S, p, s_0)$ be the $W$-TM constructing $f$. Assume $\text{card}(V) = n$, $\text{card}(S) = m$. Then, we can reach in one step starting from an elementary description at most $(n + 2)(m + 1) = k$ elementary descriptions. Therefore, starting from $1/(s_0, x\#)$, and halting after $r$ steps, the final configuration will have at most $k^r$ elementary descriptions with nonzero degree. This is a finite number and hence $Im(f, x)$ is a finite set. According to our definitions, each final tape content can be interpreted as an output, hence $Im(f, x)$ is nonempty. □
Gerla [5] defines a hierarchy of recursive $W$-functions. We will prove now that $FR$ is strictly included in the bottom class of the hierarchy.

**Definition 11** (Gerla [5]). A $W$-function $f$ from $I^*$ to $O^*$ is partially recursive (belongs to $PR$ class) if every cut of $f$ is a recursively enumerable subset of $I^* \times O^*$. Moreover $f$ is strongly partially recursive (belongs to $SPR$ class) if the restriction of $f$ to its support $\{(x,y): f(x,y) \neq 0\}$ is a partially recursive map from $I^* \times O^*$ to $W$. Finally $f$ is with recursive graph (belongs to class $WRG$) if it is a recursive map from $I^* \times O^*$ to $W$.

**Theorem 1** (Gerla [5]). $WRG \subseteq SPR \subseteq PR$ and the inclusions are strict.

**Proposition 2.** $FR \subseteq WRG$.

**Proof (Sketch).** Let $f$ be a fin-recursive $W$-function. There exists a $W$-TM $T$ which constructs $f$. It is proved in [5] that it is possible to define a gödelization in the set of $W$-TMs. Note that the gödelization of $T$ encodes the set of all its 5-uples (state, symbol, action, state, degree). Let $t$ be the number of $T$.

We must prove that $f : I^* \times O^* \rightarrow W$ is recursive. So, we must define a crisp Turing machine $T'$ such that it yields the degree $f(x,y)$ as output when the input is the string $x\#y\#t$. Then $T'$ goes on and, by decoding $t$, carries out every possible computation in one step that would be carried out by $T$, while storing in another tape (the degree tape) the corresponding degrees also decoded from $t$. Immediately $T'$ carries out every possible computation of $T$ in exactly two steps, and so on, by using the well known “dovetailing” method. Since every computation of $T$ halts, this process finishes in a finite number of steps. After the termination of the process, $T'$ has stored in the tapes both the outputs of $T$ $z_1, \ldots, z_n$ and the associated degrees. Then $T'$ compares every output $z_i$ with the input $y$, if any $z_i$ matches, $T'$ outputs the associated degree, otherwise $T'$ outputs 0. It is an easy, although cumbersome task, to define $T'$ with total formality. Hence $f(x,y)$ is (crisp) computable and $f$ is a $W$-function of the class $WRG$. ⊓⊔

Furthermore, the inclusion is strict, as shown by the following example:

**Example 2.** Let us consider the $W$-function $f : I^* \times I^* \rightarrow W$ defined as follows:

$$f(x,y) = \begin{cases} l_0 & \text{if } \text{dist}(x,y) \text{ is even,} \\ l_1 & \text{if } \text{dist}(x,y) \text{ is odd,} \end{cases}$$

where dist$(x,y)$ is the distance according to the lexicographic order. Obviously, $f \in WRG$. However, $f$ is not fin-recursive, since for each $x \in I^*$, Im$(f,x)$ is not finite (Proposition 1).

We now study the conditions for a $WRG$ function to belong to $FR$. First notice that the values $f(x,y) \in \{l_0, \ldots, l_r\}$ can be easily codified by a finite set $V_d$ of additional symbols not in $V$, $V_d = \{x_0, \ldots, x_r\}$. Let us call $V' = V \cup V_d$.

**Definition 12.** Let $f$ be a $W$-function from $I^*$ to $O^*$. We define the (crisp) function $s_f : I^* \rightarrow O^*$ in the following way: if Im$(f,x)$ is finite, then $s_f(x) = \sup(f,x) = \sup(\text{Im}(f,x))$, where $\sup(A)$ is the least upper bound of $A$ with respect to the lexicographic order; otherwise, $s_f(x)$ remains undefined.

**Proposition 3.** Let $f$ be a $W$-function from $I^*$ to $O^*$. Then, if $f \in WRG$ and $s_f$ is total (crisp) recursive, then $f \in FR$. 

Proof (Sketch). Since \( s_f \) is total, \( \text{Im}(f, x) \) is finite for every \( x \). Assume \( f \in \text{WRG} \). Then the crisp function \( f : I^* \times O^* \rightarrow W \) is totally recursive and hence there exists a (crisp) Turing machine \( M_1 \) that computes \( f \) for every input \((x, y)\). Assume \( s_f \) is (crisp) total recursive. Then there exists another (crisp) Turing machine \( M_2 \) that computes \( \sup(\text{Im}(f, x)) \) for every input \( x \). We have to show a (fuzzy) \( W\)-TM \( M \) that computes \( \text{Im}(f, x) \) with the associated degrees for every input \( x \). First we define an auxiliary (crisp) Turing machine \( M' \) that, given \( x \) as input, outputs the finite sequence of pairs \((y, f(x, y))\) for every \( y \in \text{Im}(f, x) \). By suitably modifying \( M' \), we will obtain the desired \( M \).

Construction of \( M' = (V', S', p', s_0) \): \( M_2 \) is used with input \( x \in I^* \) thus obtaining \( \sup(f, x) \). Then a machine \( M'' \) generates sequentially all elements \( y \in V^* \) from \( \varepsilon \) to \( \sup(f, x) \) in the lexicographic order, and writes them next to \( x \). For every written pair \( x \# y \) a suitable version of the machine \( M_1 \) is applied thus obtaining \( y \# f(x, y) \). The three machines \( M_2, M'', M_1 \) together define the machine \( M' \).

\( M \) is obtained from \( M' \) as follows: \( M = (V, S, p, s_0) \), \( V = V', S = S' \cup \{Q, Q'\} \) where \( Q, Q' \notin S', p \) is obtained from \( p' \) by changing the state \( h \) into the new state \( Q \) and adding the following transitions:

\[
\begin{align*}
p(Q, \#, L, Q) &= 1, \\
p(Q, x_0, \#, Q') &= l_0, \\
&\quad \ldots \\
p(Q, x_r, \#, Q') &= l_r, \\
p(Q', \#, L, h) &= 1.
\end{align*}
\]

In this way, \( M \) obtains the description \( f(x, y)/(h, y\#) \) from the previously computed description \( 1/(Q, y \# f(x, y)\#) \); to sum up, \( M \) outputs every \( y \in \text{Im}(f, x) \) with degree \( f(x, y) \). It is again an easy, although cumbersome task, to define \( M \) with total formality. \( M \) is the \( W\)-TM that constructs \( f \), and \( f \) is fin-recursive.

Since it is obvious that for every \( f \in \text{FR} \), \( s_f \) is totally recursive, these two propositions can be merged in the following theorem:

**Theorem 2.** For every \( f \in \text{WRG} \), \( f \in \text{FR} \) iff \( s_f \) is recursive.

3. The language \( XL \)

3.1. A short description of \( XL \)

The language \( XL \) \([10,12]\) is a simple programming language whose data are fuzzy subsets of \((Z^+)^n\). In short, the language can be described as follows: there are two crisp assignments, namely \( X := 0 \) and \( \text{succ}(Y) \) and a set of fuzzy assignment \( X := \text{FUZZ}_n(Y) \), where \( X, Y \) are variables. A \( W \)-function \( f_n : Z^+ \rightarrow Z^+ \) is associated with each fuzzy assignment label \( \text{FUZZ}_n \). There is a main control structure, namely an indefinite loop \( \text{WHILE} \ X \neq 0 \ \text{DO} \ldots \text{DO} \). \( X \), as usual, is called the control variable of the loop. It is well known (see, for example, [16]) that conditional control (IF) and definite loops (FOR) can be expressed in terms of indefinite loops. The meaning of an \( XL \) program is defined in \([10,12]\) in terms of its operational semantics, i.e., considering a program as a device that computes a function \( \Omega_n(a, b) \). An \( XL \) program has two sets of distinguished variables – input variables and output variables – and computes a \( W \)-function from input into output variables.

Let us describe in a less formal way how a program is fuzzily executed and the concept of configuration. Due to fuzzy inputs or assignments, a program can reach a point from which every instruction must be executed in a fuzzy way; this is specially important when considering loops or conditionals. When the program is
completed, a set of outputs is obtained, and there is a degree associated to every output; i.e., the output of a program is a fuzzy subset. The execution of a – crisp-program could be viewed as the generation of a sequence of “crisp configurations”. Every configuration represents the set of pairs \((variable, value)\) in a given instant of the execution. When considering XL programs, this sequence is not defined in a crisp way; in fact, arising from a certain point, there are several computing branches with an associated degree for each branch. In this way, the execution of an XL program could be viewed as the generation of a sequence of fuzzy configurations, where a fuzzy configuration is a fuzzy set of crisp configurations. In the following we will call “configuration element” to these crisp configurations and reserve the name “configuration” for fuzzy configurations.

The semantics of a WHILE loop, informally speaking, could be described as follows.

Let \(X\) be the control variable of an indefinite loop. For each configuration element with \(X \neq 0\), the body of the loop is executed. Results are grouped together with elements with \(X = 0\), giving an intermediate configuration. The process is iterated until every element in the configuration has \(X = 0\). If such a configuration is not reached, the process goes on forever. Notice that this fuzzy loop, when applied to crisp data, is equivalent to a crisp loop.

In \([10, 12]\) it was proved that every XL program without indefinite loops defines a total \(W\)-function, in the sense that for every initial configuration exactly one final configuration is reached. However, the situation is different in the presence of indefinite loops; in the general case there are nonterminating computations, and no final configuration is reached from a certain set of initial data.

More formally, let \(H\) be a set of recursive \(W\)-functions, \(H \subseteq PR\). A \(W\)-function \(f\) is XL\((H)\)-programmable iff there exists an XL-program \(P\) that computes \(f\), provided that every assignment in \(P, X := FUZZ_n(Y)\), is associated to a \(W\)-function \(f_n \in H\). The class of XL\((H)\)-programmable functions will be denoted \(F(XL(H))\). Since a programmable function can be partial, we also define the set of total programmable functions \(FT(XL(H))\).

### 3.2. Multitape W-TMs and XL-programs

In this section we study the concept of multitape \(W\)-TM. Informally speaking, such a machine reads from and writes on several tapes. This construct is well known in crisp computability. The concept developed in this section will be used in Section 4.

**Definition 13.** A \(k\)-tape \(W\)-TM is a \(4\)-uple \(Z = (V, S, p, s_0)\) where \(V\) and \(S\) are finite nonempty sets, \(V \cap S = \emptyset; s_0 \in S\) and \(p\) is a \(W\)-function from \(S \times (V)^k\) to \((V \cup \{L, R\})^k \times (S \cup \{h\}); L, R \notin V, h \notin S\).

An elementary description of \(Z\) is a \(k+1\)-tuple \((q, u_1a_1v_1, \ldots, u_ka_kv_k)\) where \(q \in S \cup \{h\}\) and \(u_ia_i \in (\{e\} \cup (V - \{\#\}))^* (V^* (V - \{\#\}) \cup \{e\})\).

An elementary description is final when \(q = h\).

The concepts of description, initial description, final description, computation and halting are defined for multitape \(W\)-TM in a way analogous to the case of one-tape \(W\)-TM, as seen in Section 2.

**Definition 14.** Let \(f\) be a \(W\)-function from \((I^*)^\gamma\) to \((O^*)^\delta\). We say that the \(W\)-TM \(T = (V, S, p, s_0)\) constructs \(f\) iff

(a) \(I, O \subseteq V\).

(b) For each \(x \in (I^*)^\gamma\), \(T\) halts with input \(x\) and the final description reached is \(\Sigma\) such that \(f(x, y) = \Sigma \mu_\Sigma(d)\), where \(\Sigma\) represents the application of \(\oplus\) to the set of all elementary descriptions \(d\) such that \(\mu_\Sigma(d) \neq 0, d = (h, c_1, \ldots, c_k)\) and \(c_i = y_i \#\), \(i = 1, \ldots, s\), \(y = (y_1, \ldots, y_s)\).

A \(W\)-function \(f\) is fin-recursive iff there exists a \(W\)-TM that constructs \(f\).
FR(r, s) will denote the class of all fin-recursive $W$-functions from $(\mathbb{Z}^+)^r$ to $(\mathbb{Z}^+)^s$. Notice that these definitions are consistent with those given for a (one-tape) $W$-TM in Section 4.1. FR will be in the following the union of every FR(r, s) $(r, s \in \mathbb{Z}^+, s > 0)$.

A natural mapping can be established between XL-programs and multitape machines. This idea is developed in the rest of the section. Obviously, we will focus on machines with alphabet $V = \{|, \#\}$, i.e., machines computing the class FR(r, s).

**Notation.** Let C be $C = (c_1, \ldots, c_k) \in (\mathbb{Z}^+)^k$. The pair $(q, C)$ will stand for an elementary description of a $k$-tape $W$-TM $(q, C_1', \#^k, C_2', \#^k, \ldots, C_k', \#^k)$ where $C_i' = | \ldots (c_i$ times).

**Definition 15.** Let C be a configuration of a $k$-program in XL and let $A$ be a description of a $k$-tape $W$-TM. C and $A$ are equivalent iff there exists a finite subset $\{b_1, \ldots, b_n\} \subset (\mathbb{Z}^+)^k$, a sequence $(x_1, \ldots, x_n)$ in $W$, and a state $q$ such that

$$C = x_1/b_1 + x_2/b_2 + \cdots + x_n/b_n \quad \text{and} \quad A = x_1/(q, b_1) + x_2/(q, b_2) + \cdots + x_n/(q, b_n).$$

**Definition 16.** Let $P$ be a $k$-program in XL and let $M$ be a $k$-tape $W$-TM, $M = (V, S, P, s_0)$. $P$ and $M$ are equivalent iff for each $a, b \in (\mathbb{Z}^+)^k$ either (i) or (ii) holds:

(i) $\Omega P(a, b)$ is not defined and $M$ does not halt with input $a$.
(ii) $\Omega P(a, b) = \beta$ and $M$ halts with input $a$ reaching a final description $A$ such that $\mu_A(b) = \beta$.

**Comment.** If $P$ and $M$ are equivalent, then there exists a one-to-one mapping between the pairs (initial description, final description) in $M$ and the pairs (initial configuration, final configuration) in the derivation tree of $P$. In other words, if the initial configuration of $P$ and the initial description of $M$ are equivalent, then both $P$ and $M$ either engage in a nonterminating computation or halt, and, if both $P$ and $M$ halt, then the final configuration of $P$ and the final description of $M$ are equivalent.

### 4. Programming characterization of FR

**Definition 17.** Let $I_1, \ldots, I_r$ and $\psi_{01}$ be the following $W$-functions from $\mathbb{Z}^+$ to $\mathbb{Z}^+$:

$$I_i(x, y) = \begin{cases} 1 & \text{if } y = x, \\ 0 & \text{otherwise,} \end{cases} \quad \psi_{01}(x, y) = \begin{cases} 1 & \text{if } (y = 0) \text{ or } (y = 1), \\ 0 & \text{otherwise,} \end{cases}$$

$i = 1, \ldots, r$.

We define the family $C = \{I_1, \ldots, I_r, \psi_{01}\}$. Notice that $C \subset FR$.

For the proofs to be developed in this section, we need a theorem stated and proved in [16] whose roots may be found in [17,2] (see, for example, [13]). With the notation used in this paper, the theorem can be stated as follows:

**Theorem 3** (Sommerhalder and van Westrhenen [16]). A (crisp) function $f$ is (crisp) recursive iff $f$ is XL($(I_i)$)-programmable.

**Theorem 4.** Let $f$ be a fin-recursive $W$-function from $\mathbb{Z}^+$ to $\mathbb{Z}^+$. Then, $f$ is XL($C$)-programmable.
Proof.

\begin{verbatim}
P3;
begin
    X5 := 0;
    X2 := P2 (X1);
    REPEAT X2 TIMES
        IF X5 \neq 1 THEN
            X4 := P1 (X1, X3);
            IF X4 \neq 0 THEN
                X5 := FUZZ0 (X4);
                IF X5 = 1 THEN
                    X6 := FLATTEN (X3, X4)
                FI
            FI
        FI
    SEMIT;
    IF X5 \neq 1 THEN
        X4 := P1 (X1, X3);
        X6 := FLATTEN (X3, X4)
    FI
    X1 := X6
end
\end{verbatim}

Program P3.

Let $f$ be a \textit{fin}-recursive $W$-function from $\mathbb{Z}^+$ to $\mathbb{Z}^+$; then, by Proposition 2, $f \in WRG$, and by Definition 11, $f$ is a recursive total function from $\mathbb{Z}^+ \times \mathbb{Z}^+$ to $W$, and by Theorem 3 there exists an $XL(\{I_1\})$ (crisp) program that computes $f$. Let $P_1 = (2, 1, Q_1)$ be this program; $P_1$ is also an $XL(C)$ program. By considering that the identifier of this program is $P_1$, this program can be used as a macro in the following way:

$$\Xi_i := P_1(\Xi_j, \Xi_k)$$

whose execution yields the following result: given $a, b \in \mathbb{Z}^+$, values for $\Xi_j$ and $\Xi_k$ respectively, $\Xi_i$ stores the degree $f(a, b)$-encoded by a positive integer $0, \ldots, r$.

On the other hand, the function $s_f : \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ is (crisp) recursive and there exists a $XL(\{I_1\})$ program that computes $s_f$. Let $P_2 = (1, 1, Q_2)$ be this program in $XL(\{I_1\})$ and also in $XL(C)$. Let its identifier be $P_2$.

By using these macros, it is possible to write an $XL(C)$-program $P_3 = (1, 1, Q_3)$ that computes $f$. (Macro expansion is fully described in [9]).

In this program, the function associated to $FUZZ0$ is $\psi_{01}$. The macro $FLATTEN$ is associated to the program $P_4 = (2, 1, Q)$ whose code $Q$ is:
FLATTEN;
begin
  IF X2=1 THEN X1:=FUZZ1(X1) FI;
  IF X2=2 THEN X1:=FUZZ2(X1) FI;
  ............
  IF X2=r THEN X1:=FUZZr(X1) FI
end

Program $P_4$.

The $W$-functions associated to FUZZ$i$ ($i = 1,\ldots,r$) are $I_i$.

A formal proof of the theorem could be developed. This proof would use the semantic definitions of attributes \textit{coin} and \textit{cof} in every node of the derivation tree of $P3$ [9,12]. However, this proof would be tedious and hardly readable. For this reason, the proof presented here is not fully formalized. The proof will proceed by showing the configurations at the beginning, at an intermediate point, and at the end of the computation of $P3$. Notice first the role played by each variable in $P3$:

- $X1$: stores the input and the output (line 18).
- $X2$: stores the value of $s_f(X1)$ and controls the main loop (2–13).
- $X3$: takes all the values from 0 to $s_f(X1)$; those with $f(X1,X3) \neq 0$ will be the output.
- $X4$: stores the degree $f(X1,X3)$ encoded as a positive integer.
- $X5$: stores either 0 or 1 to control a binary branching.
- $X6$: auxiliary store for the outputs.

Let $a$ be the input. The initial configuration is $1=(a,0,0,0,0,0)$. Line 1 stores in $X2$ the greatest integer $b$ such that $f(a,b) \neq 0$ and for every $b' > b$, $f(a,b') = 0$. In fact, the utility of $b$ is to control the iterations of the main loop, by ensuring that the last value will be $b$. Notice that the degree corresponding to the computation with the value $b$ is not 0.

Now the program must obtain the values $b_1 < b_2 < \cdots < b_s = b$ with degrees $x_1, x_2, \ldots, x_s$ such that $f(a,b_i) = x_i \in \{l_1,\ldots,l_r\}$.

This task is performed by the loop in lines 2–13. Inside the loop, if $b' < b_1$ then $f(a,b') = 0$, hence the macro in line 4 makes $X4=0$ and no action is carried out in the statements embraced by the IF in line 5. Therefore the configuration is not changed.

Let us assume that there have been computed and stored in $X6$ some values $b_i: b_1,\ldots,b_{j-1}$. After executing $b_{j-1}$ times the main loop, the resulting configuration will be

$$x_1/(a,b,b_j,n_1,1,b_1) + \cdots + x_{j-1}/(a,b,b_j,n_{j-1},1,b_{j-1}) + 1/(a,b,b_j,n_{j-1},0,0).$$

Notice that $X5=1$, except for the last elementary configuration. Then the first $j-1$ elementary configurations never satisfy the condition in line 3, because the value of $X5$ is not changed outside the IF statements.

Therefore, the only modified configuration is the last one. Let us describe a complete execution of the main loop with respect to the changed elementary configurations.

After the execution of line 4 the configuration is

$$1/(a,b,b_j,n_j,0,0),$$

where $n_j \neq 0$ stands for degree $x_j$. 
Then the condition in 5 is satisfied and after 6 it is
\[
1/(a, b, b_j, n_j, 1, 0) + 1/(a, b, b_j, n_j, 0, 0).
\]

The condition in 7 is only satisfied by the first elementary configuration; after line 8 the configuration is
\[
z_j/(a, b, b_j, n_j, 1, b_j) + 1/(a, b, b_j, n_j, 0, 0).
\]

Now, the first elementary configuration must remain unchanged, since \(X5=1\); the other can be eventually modified. To sum up, the complete configuration is now
\[
z_1/(a, b, b_j, n_1, 1, b_1) + \cdots + z_{s-1}/(a, b, b_j, n_{s-1}, 1, b_{s-1}) + z_j/(a, b, b_j, n_j, 1, b_j) + 1/(a, b, b_j, n_j, 0, 0),
\]
where a new partial result has appeared.

Let us study now the last step of the execution for \(b_y=b\). At this moment, the loop has been executed \(b\) times for the values of \(X3\), \(0,1,\ldots,b-1\). Now \(X3=b\), and \(f(a, b) \neq 0\). The loop is finished and the configuration is
\[
z_1/(a, b, b, n_1, 1, b_1) + \cdots + z_{s-1}/(a, b, b, n_{s-1}, 1, b_{s-1}) + 1/(a, b, b, n_{s-1}, 0, 0).
\]

Since only the last elementary configuration has \(X5 \neq 1\), only the last elementary configuration will be modified. After 15 it is
\[
1/(a, b, b, n_s, 0, 0)
\]
and after 16:
\[
z_s/(a, b, b, n_s, 0, b) \quad (b = b_y).
\]

Finally, after 18 the configuration is
\[
z_1/(b_1, b, b, n_1, 1, b_1) + \cdots + z_{s-1}/(b_{s-1}, b, b, n_{s-1}, 1, b_{s-1}) + z_s/(b_s, b, b, n_s, 0, b_s).
\]

The semantic \(W\)-function associated to \(P\) is \(\phi_p : \mathbb{Z}^+ \times \mathbb{Z}^+ \to W\) such that for \(i=1,\ldots,s\) \(\phi_p(a, b_i) = z_i\) and for any other \(b\), \(\phi_p(a, b) = 0\), so \(\phi_p = f\) and hence \(f\) is \(XL(G)\)-programmable. \(\square\)

**Example 3.** Let \(f\) be a fin-recursive \(W\)-function \(f : \mathbb{Z}^+ \times \mathbb{Z}^+ \to W\) such that for argument 9, the images are as follows: \(f(9, 0) = 0\), \(f(9, 1) = 0\), \(f(9, 2) = l_4\), \(f(9, 3) = l_6\), \(f(9, 4) = l_1\), \(f(9, 5) = 0\), \(f(9, 6) = l_3\), for every \(x > 6\), \(f(9, x) = 0\). Let us consider the program \(P\) proposed in the above proof. We will show, for example, that \(\phi_p(9) = f(9)\).

The initial configuration is: \(1/(9, 0, 0, 0, 0, 0)\)

After the execution of 1, the configuration is \(1/(9, 6, 0, 0, 0, 0)\).

Now, the definite loop (2–13) is iterated 6 times for \(X3=0,\ldots,5\) (instruction 12). The final value of \(X3\) is 6.

According to 3, \(X5 \neq 1\), hence 4 is executed; since \(f(9, 0) = 0\), the configuration is not changed. Since \(X4=0\), the next statement must be 12, whose execution yields
\[
1/(9, 6, 1, 0, 0, 0)
\]
and we come back to 3. Since \(f(9, 1) = 0\), the process is iterated and after 12 we have
\[
1/(9, 6, 2, 0, 0, 0).
\]

Again in 3, it holds that \(X5 \neq 1\), therefore 4 is executed, so computing for \(X4\) the value 4 (encoding the degree \(l_3\)), and the configuration is
\[
1/(9, 6, 2, 4, 0, 0).
\]
Now, in 5 it holds that \( X_4 \neq 0 \), line 6 must be executed and it is
\[
1/(9, 6, 2, 4, 1, 0) + 1/(9, 6, 2, 4, 0, 0).
\]
The condition of 7 (\( X_5 = 1 \)) is satisfied by the first elementary configuration and after the execution of 8 it is
\[
I_4/(9, 6, 2, 4, 1, 2) + 1/(9, 6, 2, 4, 0, 0).
\]
Only \( X_3 \) will change in the first elementary configuration; however, the degree and value of \( X_6 \) remain unchanged. Now it is the second elementary configuration which is essentially changed. After 12 we have
\[
I_4/(9, 6, 3, 4, 1, 2) + 1/(9, 6, 3, 4, 0, 0).
\]
For \( X_5 \neq 1 \), after the execution of 4, \( X_4 \) stores the value 6 (encoding the degree \( l_6 \)), and we have
\[
I_4/(9, 6, 3, 4, 1, 2) + 1/(9, 6, 3, 6, 0, 0)
\]
\( X_4 \neq 0 \), therefore after executing 6 we have
\[
I_4/(9, 6, 3, 4, 1, 2) + 1/(9, 6, 3, 6, 1, 0) + 1/(9, 6, 3, 6, 0, 0).
\]
For the second elementary configuration \( X_5 = 1 \), therefore line 8 is executed and we have
\[
I_4/(9, 6, 3, 4, 1, 2) + l_6/(9, 6, 3, 6, 1, 3) + 1/(9, 6, 3, 6, 0, 0).
\]
After the following execution of the loop we have
\[
I_4/(9, 6, 4, 4, 1, 2) + l_6/(9, 6, 4, 6, 1, 3) + l_1/(9, 6, 4, 1, 1, 4) + 1/(9, 6, 4, 1, 0, 0).
\]
Now \( X_3 \) is incremented in 12, and we have
\[
I_4/(9, 6, 5, 4, 1, 2) + l_6/(9, 6, 5, 6, 1, 3) + l_1/(9, 6, 5, 1, 1, 4) + 1/(9, 6, 5, 1, 0, 0).
\]
Only the last elementary configuration satisfies \( X_5 \neq 1 \). After 4 we have
\[
I_4/(9, 6, 5, 4, 1, 2) + l_6/(9, 6, 5, 6, 1, 3) + l_1/(9, 6, 5, 1, 1, 4) + 1/(9, 6, 5, 0, 0, 0)
\]
and \( X_4 \neq 0 \) does not hold; line 12 is executed and the configuration becomes
\[
I_4/(9, 6, 6, 4, 1, 2) + l_6/(9, 6, 6, 6, 1, 3) + l_1/(9, 6, 6, 1, 1, 4) + 1/(9, 6, 6, 0, 0, 0).
\]
The main loop (2–13) has been executed 6 times; in line 14 it holds that \( X_5 \neq 1 \) in the last elementary configuration. Line 15 is executed and we have
\[
I_4/(9, 6, 6, 4, 1, 2) + l_6/(9, 6, 6, 6, 1, 3) + l_1/(9, 6, 6, 1, 1, 4) + 1/(9, 6, 6, 3, 0, 0).
\]
After 16 we obtain
\[
I_4/(9, 6, 6, 4, 1, 2) + l_6/(9, 6, 6, 6, 1, 3) + l_1/(9, 6, 6, 1, 1, 4) + l_3/(9, 6, 6, 3, 0, 6).
\]
Finally, after 18 the final configuration becomes
\[
I_4/(2, 6, 6, 4, 1, 2) + l_6/(3, 6, 6, 6, 1, 3) + l_1/(4, 6, 6, 1, 1, 4) + l_3/(6, 6, 6, 3, 0, 6).
\]
Since \( P \) has only one output variable \( (X_1) \), the output is
\[
\]
Therefore \( \phi_P(9, 2) = I_4, \phi_P(9, 3) = l_6, \phi_P(9, 4) = l_1, \phi_P(9, 6) = l_3; \) otherwise \( \phi_P(9, x) = 0; \) hence it yields the same values as \( f \) for the first argument equal to 9.
Corollary 1. $FR \subseteq FT(XL(C))$.

Proof. Let $f$ be an element of $FR$, i.e. $f$ is a $\text{fin-recursive } W$-function from $(\mathbb{Z}^+)^n$ to $(\mathbb{Z}^+)^k$. Let us consider Cantor’s encodings (see for example [16]) $c_n: (\mathbb{Z}^+)^n \to \mathbb{Z}^+$ and $c_k: (\mathbb{Z}^+)^k \to \mathbb{Z}^+$. Both these encodings and their inverses are bijective crisp recursive functions. Then we can consider $f'$ from $\mathbb{Z}^+$ to $\mathbb{Z}^+$ defined by $f'(x, y) = f(c_n^{-1}(x), c_k^{-1}(y))$. Obviously, $f'$ is a $\text{fin-recursive } W$-function and by Theorem 4, $f'$ is $XL(C)$-programmable, hence $f(u, v) = f'(c_n(u), c_k(v))$ is also $XL(C)$-programmable, i.e., $f$ is an element of $F(XL(C))$. Since every $f \in FR$ is total, then $f \in FT(XL(C))$. □

In the following, we will prove the reverse inclusion, i.e., that $FT(XL(C)) \subseteq FR$. The proof will make use of multitape $W$-TM’s. For every program $P$, we will show (Lemma 4) an equivalent machine $M$, in the sense of Section 3.2. The proof of Lemma 4 will be developed by structural induction on the program $P$, beginning from one statements assignment programs (Lemma 3), and simulating sequence (Lemma 1) and nesting of statements (Lemma 2).

Lemma 1. Let $P_1$ be a $k$-program in XL with code $P_1$; begin B1 end. Let $P_2$ be a $k$-program in XL with code $P_2$; begin B2 end. Assume there exist two $k$-tape machines $T_1$ and $T_2$ equivalent, respectively, to $P_1$ and $P_2$.

Then there exists a $k$-tape machine $T$ equivalent to the XL-program $P$ whose code is $P_12$; begin B1; B2 end

Proof. Let $T_i = (V, S_i, p_i, s_{0i})$ ($i = 1, 2$) be the $W$-$MT$ equivalent to the program $P_i$.

Then the $k$-tape $W$-$MT$ $T = (V, S, p, s_{0})$ is defined as follows:

$$S = S_1 \cup S_2,$$

$$p(q, C, C', q') = p_1(q, C, C', q') \quad \text{if } q.q' \in S_1 \text{ and } q' \neq h,$$

$$p(q, C, C', s_{02}) = p_1(q, C, C', h) \quad \text{if } q \in S_1,$$

$$p(q, C, C', q') = p_2(q, C, C', q') \quad \text{if } q.q' \in S_2,$$

$$s_0 = s_{01}$$

The transitions of $T$ are, roughly speaking, those of $T_1$ plus those of $T_2$: however, in the halting transitions of $T_1$, $h$ is replaced by $s_{02}$. $T$ starts from the initial state of $T_1$; operates as $T_1$ would operate until $T_1$ would halt (so simulating $S1$); then operates as $T_2$ would operate (so simulating $S2$). It is easy to verify that $T$ is equivalent to $P$. □

In the following, the machine $T$ so defined will be represented as $T_1oT_2$ ($T_1$ concatenated to $T_2$).

Lemma 2. Let $P_1$ be a $k$-program in XL with code $P_1$; begin B1 end. Assume there exists a $k$-tape machine $T_1$ equivalent to $P_1$. Then there exists a $k$-tape machine $T$ equivalent to the $k$-program $P$ whose code is $P_{11}$; begin while $X_i \neq 0$ do B1 od end

Proof. Let $T_1 = (V, S_1, p_1, s_{01})$ be the $W$-$MT$ equivalent to the program $P_1$.

Then the $k$-tape $W$-$MT$ $T = (V, S, p, s_{0})$ is defined as follows:

$$S = S_1 \cup \{s_0, s'\}, \quad (s_0, s' \notin S_1),$$

$$p(s_0, \#(i), L(i), s') = 1,$$

$$p(s', |(i), R(i), s_{01}) = 1,$$

$$p(s', \#(i), R(i), h) = 1,$$
\[ p(q,C,C',q') = p_1(q,C,C',q') \quad \text{if } q,q' \in S_1 \text{ and } q' \neq h, \]
\[ p(q,C,C',s_0) = p_1(q,C,C',h) \quad \text{if } q \in S_1. \]

That is to say, \( T \) verifies if the content of the \( i \)th tape is \( \neq 0 \); in this case, the state of \( T \) becomes the state \( s_{01} \) – initial state for \( T_1 \) – and \( T \) operates as \( T_1 \) would operate until \( T_1 \) would halt, the only difference being the substitution of the halting state \( h \) by the initial state \( s_0 \); otherwise, \( T \) halts. Obviously, \( T \) and \( P \) are equivalents: \( T \) terminates iff \( P \) halts and, if \( P \) halts, the final description reached by \( T \) is equivalent to the final configuration of \( P \) after the execution of \textbf{begin while} \( X_i \neq 0 \) \textbf{do} \( B_1 \) \textbf{od} \. \quad \square 

In the following, the machine \( T \) so defined will be represented by \( w(T,i) \).

**Lemma 3.** For every program \( P \) without loops, there exists an equivalent \( W \)-TM.

**Proof.** We first consider one-statement programs, and then we will consider programs with sequences of statements.

Let us assume that \( P \) is

\[ \text{ident; begin (assignment) end} \]

and \( \langle \text{assignment} \rangle \) is \( X_i := 0; \)

\( T \) is equivalent to \( P: T = (V,S,p,s_0), S = \{s_0, s_1, s_2\} \), and \( p \) is defined as follows:

\[ p(s_0,\#(i),L(i),s_1) = 1 \]
\[ p(s_1,(i),\#(i),s_2) = 1 \]
\[ p(s_1,\#(i),\#(i),h) = 1 \]
\[ p(s_2,\#(i),L(i),s_1) = 1 \]

Assume now that \( \langle \text{assignment} \rangle \) is \( X_i := \text{suc}(X_j); \) the machine \( T \) defined below is equivalent to the program \( P: \)

\[ T = (V,S,p,s_0), S = \{s_0,\ldots,s_6\} \]
\[ p(s_0,\#(i),L(i),s_1) = 1 \]
\[ p(s_1,\#(i),\#(i),s_2) = 1 \]
\[ p(s_1,\#(i),\#(i),s_3) = 1 \]
\[ p(s_2,\#(i),L(i),s_1) = 1 \]
\[ p(s_3,(j),L(j),s_4) = 1 \]
\[ p(s_4,(j),L(j),s_5) = 1 \]
\[ p(s_4,\#(j),L(j),s_4) = 1 \]
\[ p(s_5,(j),L(j),s_5) = 1 \]
\[ p(s_5,\#(j),\#(i),s_6) = 1 \]
\[ p(s_6,\#(j),\#(i),s_6) = 1 \]
\[ p(s_6,\#(j),R(i,j),s_5) = 1 \]
\[ p(s_6,(i),R(i,h) = 1 \]
The first 4 lines erase the \(i\)th tape. Transitions with states \(s_3\) and \(s_4\) put the machine at the beginning of content \(a_i\) of the \(i\)th tape. Transitions with states \(s_5\) and \(s_6\) copy the content of the \(j\)th tape onto the \(i\)th tape. Finally, another “|” is written on the \(i\)th tape.

Assume now that \(\langle\text{assignment}\rangle\) is \(X_i := \text{FUZZ}_0(X_j)\). Remember that the function associated to \(\text{FUZZ}_0\) is \(\psi_{01}\). The following machine \(T\) is equivalent to \(P\):

\[
T = (V, S, p, s_0), \quad S = \{s_0, \ldots, s_3\},
\]
\[
p(s_0, #(i), L(i), s_1) = 1,
\]
\[
p(s_1, (i), #(i), s_2) = 1,
\]
\[
p(s_1, #(i), #(i), h) = 1,
\]
\[
p(s_1, (i), (i), s_3) = 1,
\]
\[
p(s_2, #(i), L(i), s_1) = 1,
\]
\[
p(s_3, (i), R(i), h) = 1.
\]

The \(W\)-function \(\psi_{01}\) always yields the images 0 (degree 1) and 1 (degree 1). In fact, \(T\) erases the \(i\)-th tape and then (a) halts with degree 1 and (b) writes a single “|” and halts with degree 1.

Assume now that \(\langle\text{assignment}\rangle\) is \(X_i := \text{FUZZ}_k(X_j)\) \((k = 1, \ldots, r)\). Remember that the \(W\)-function associated to \(\text{FUZZ}_k\) is \(I_k\). The following machine \(T\) is equivalent to \(P\):

\[
T = (V, S, p, s_0), \quad S = \{s_0, \ldots, s_6\},
\]
\[
p(s_0, #(i), L(i), s_1) = 1,
\]
\[
p(s_1, (i), #(i), s_2) = 1,
\]
\[
p(s_1, #(i), #(i), s_3) = 1,
\]
\[
p(s_2, #(i), L(i), s_1) = 1,
\]
\[
p(s_3, (j), L(j), s_4) = 1,
\]
\[
p(s_4, (j), R(j), s_5) = 1,
\]
\[
p(s_4, (j), L(j), s_4) = 1,
\]
\[
p(s_5, (j), (i), s_6) = 1,
\]
\[
p(s_5, (j), #(i), h) = I_k,
\]
\[
p(s_6, (j), R(i, j), s_5) = 1.
\]

The \(W\)-function \(I_k\) yields \(x\) (degree \(I_k\)) with input \(x\). In fact, the only actions performed by \(T\) are copying \(x_j\) in the \(i\)th tape and halting with degree \(I_k\).

Let us now assume that \(P\) is

\[
\langle\text{ident}; \text{begin}\langle\text{assignment}\rangle; \ldots \langle\text{assignment}\rangle; \text{end}\rangle
\]

Notice that it suffices to prove the lemma for sequences of two statements. But, by Lemma 1, it is proved. Hence, Lemma 3 is also proved. \(\square\)

**Lemma 4.** For each XL(C)-program there exists an equivalent \(W\)-TM.
Proof. Let us assume that the program $P$ has no nested loops. Then, by Lemma 3, there is a $W$-TM $M$ equivalent to $P$.

Let us assume that the theorem holds for every program with loops nested at $b$ or less levels. We must prove that the theorem holds for every program with loops nested at $b + 1$ levels. Let $P$ be a program with loops nested at $b + 1$ levels; then $P$ can be written as

$$\langle \text{ident} \rangle; \text{begin } S_1; S_2; \ldots; S_n \text{ end}$$

where each $S_i$ is either an assignment or a while statement of the form $\text{while } X_i \neq 0 \text{ do } Q_i \text{ od}$ and the program $P_i \langle \text{ident} \rangle$ begins $Q_i$ and ends in a program with loops nested at $b$ or less levels. If $S_i$ is an assignment then, by Lemma 3, there exists a machine $M_i$ equivalent to the program $\langle \text{ident} \rangle\text{begin } S_i \text{end}$; if $S_i$ is a while loop then, by Lemma 2 and the induction hypothesis, there also exists a machine $T_i = w(T;i)$ equivalent to the program $\langle \text{ident} \rangle\text{begin while } X_i \neq 0 \text{ do } Q_i \text{ od end}$. In any case, an argumentation similar to that of Lemma 1 shows that there exists a $W$-TM $T = T_1 \circ T_2 \circ \cdots \circ T_n$ equivalent to $P$. $\square$

Theorem 5. $FT(XL(C)) \subseteq FR$.

Proof. Let $f$ be a $W$-function in $FT(XL(C))$. By Lemma 4, there is a $W$-TM $M$ equivalent to $P$. Since $f$ is total, $P$ terminates with every input. Hence, $M$ halts in a finite number of steps with every input. Therefore, $f \in FR$. $\square$

We have

Theorem 6. $FT(XL(C)) = FR$.

5. Conclusions and future work

The results presented in this paper are part of a continuing research on the foundations of fuzzy imperative languages and fuzzy computability [7–9,11,12,14]. From the computability side, this paper introduces a new, narrower sense of the concept of a fuzzy computable function, given by the class $FR$. From the programming side, we have shown that a close relationship exists between the class $FR$ and the family of programming languages $XL(C)$ generated when the fuzzy assignment in $XL$ is constrained to be a function belonging to a certain set $C$; in fact, we have shown that $FR = FT(XL(C))$.

Future works will extend the programming characterisation to broader classes of fuzzy functions, like $WRG$. On the other hand, practical implementations of certain extensions of $XL(H)$ are currently the object of our research.

Acknowledgements

The authors would like to thank the anonymous referees whose comments greatly improved the readability of the paper.

References


