Available online at www.sciencedirect.com

SCIENCE DIRECT®

International Journal of Approximate Reasoning
38 (2005) 175–214

INTERNATIONAL JOURNAL OF APPROXIMATE REASONING

www.elsevier.com/locate/ijar

New directions in fuzzy automata

Mansoor Doostfatemeh *, Stefan C. Kremer

Department of Computing and Information Science, University of Guelph, Guelph, Ont., Canada N1G 2W1

Received 1 January 2004; received in revised form 1 July 2004; accepted 1 August 2004
Available online 18 September 2004

Abstract

Automata are the prime example of general computational systems over discrete spaces. The incorporation of fuzzy logic into automata theory resulted in fuzzy auotomata which can handle continuous spaces. Moreover, they are able to model uncertainty which is inherent in many applications. Deterministic Finite-state Automata (DFA) have been the architecture, most used in many applications, but, the increasing interest in using fuzzy logic for many new areas necessitates that the formalism of fuzzy automata be more developed and better established to fulfill implementational requirements in a well-defined manner. This need is due to the fact that despite the long history of fuzzy automata and lots of research being done on that, there are still some issues which have not been well-established and issues which need some kind of revision. In particular, we focus on membership assignment, output mapping, multi-membership resolution, and the concept of acceptance for fuzzy automata. We develop a new general definition for fuzzy automata, and based on that, develop well-defined and application-driven methodologies to establish a better ground for fuzzy automata and pave the way for forthcoming applications.

Keywords: Automata theory; General fuzzy automata; Membership assignment; Zero-weight transition; Multi-membership resolution; Output mapping; Acceptance; Conditional acceptance

* Corresponding author. Tel.: +1 519 824 4120x56404; fax: +1 519 837 0323.
E-mail address: mdoostfa@uoguelph.ca (M. Doostfatemeh).

0888-613X/$ - see front matter
doi:10.1016/j.ijar.2004.08.001

1. Introduction

Automata have a long history both in theory and application [1–7]. Automata are the prime example of general computational systems over discrete spaces [8]. Among the conventional spectrum of automata (i.e. Deterministic Finite-state Automata (DFA), Non-deterministic Finite-state Automata (NFA), Probabilistic (stochastic) Automata (PA), and Fuzzy-Finite-state Automata (FFA)), DFA have been the most applied automata to different areas [9–11]. See [12] for more applications. DFA have been shown to be an appropriate tool for modeling systems and applications which can be realized as a finite set of states (including some final states) and transitions between them depending on some input strings [1] e.g. all logic circuits from a simple AND gate to the control unit of a supercomputer. In this paper, we are focussing on Fuzzy Finite-state Automata (FFA), which incorporate fuzziness into the internal state representation and output of these computational systems. Fuzzy automata not only provide a systematic approach to handle uncertainty in such systems, but also are able to handle continuous spaces [15].

There is an increasing interest in using fuzzy logic in many new areas. Fuzzy logic [16] is a very efficient method for handling imprecision which is an intrinsic property of many systems [17]. It provides a nice systematic approach to incorporating approximate reasoning into such systems (in the way humans do) [18,19]. Moreover, fuzzy implementations of many applications are not only cheaper and faster but also make them more understandable for operators and end-users of the systems [20–27,17,28–31].

Fuzzy automata and their counterparts fuzzy grammars, combine the capabilities of automata and language theory with fuzzy logic in an elegant way [32–35]. They have been shown to be very useful for areas which are well-known to be handled by discrete mathematics and probabilistic approaches, e.g. structural matching methods [36], logical design [37]. In general, fuzzy automata provide an attractive systematic way for generalizing discrete applications [38,39,37,40,41]. Moreover, fuzzy automata are able to create capabilities which are hardly achievable by other tools [42]. On the other hand, the contribution of FFA to neural networks (more specifically recurrent NNs) has been considerable, and dynamical fuzzy systems are getting more and more popular and useful [43–46]. It seems that the demand for using FFA will increase considerably in coming years.

In spite of the long history and lots of research being done on fuzzy automata, it still seems that there are some issues which have not been well-established and issues

[1] It is noticeable that our approach is based on the key concept of *state* and the behavior of the automaton is *state-determined*. Hence, an automaton in the scope of our approach is a *discrete-time*, *discrete-state-space*, and *state-determined* machine. But, there is a more general approach where the current state and inputs (and consequently the next state) are not necessarily well-defined [8]. Instead of discrete and well-defined states and inputs, we have the probability distributions of the states and/or the inputs. This gives rise to the concepts of *hyperstates* and *hyperinputs*. However, for the approach we are taking, these are not applicable. For a more detailed discussion on the role of these concepts in modern system theory and their comparison with conventional theory see [8,13,14].

which need some kind of revision. These issues show up mostly for applicational aspects of FFA, specially when we talk about the direct representation and justification of FFA.

The rest of the paper is organized as follows: in Section 2 we present the required background, addressing some of the shortages of fuzzy automata, and present some conventions and definitions which are necessary. We also compare the realm of conventional automata emphasizing the generality of fuzzy automata. Our contributions are introduced in Section 3. They include: (1) augmented transition function, (2) multi-membership resolution, (3) output mapping, (4) analysis of the continuous operation of fuzzy automata, and (5) a new general definition for fuzzy automata (GFA: General Fuzzy Automata). In Section 4 we give some examples illustrating the capabilities achievable from these contributions. The paper ends with some concluding remarks, discussion, and future work in Section 5.

2. Background

In this section, we first define the basic concepts of fuzzy logic. Then, we show the insufficiency of the literature in this regard, establish some new terminology and summarize the conventional spectrum of automata in a comparative sense.

2.1. Fuzzy logic basics

Two basic concepts are very crucial in this paper: fuzzy sets and fuzzy power sets. There are different definitions and notations used in the literature. However, we take the approach of Klir and Yuan [17] and define these concepts as follows:

Definition 1. A fuzzy set μ_Q defined on a set Q (discrete or continuous), is a function mapping each element of Q to a unique element of the interval $[0,1]$.

$$\mu_Q : Q \to [0,1] \tag{1}$$

Then, the fuzzy power set of Q denoted as $\widetilde{P}(Q)$, is the set of all fuzzy subsets μ_Q, which can be defined on the set Q.

$$\widetilde{P}(Q) = \{\mu_Q \mid \mu_Q : Q \to [0,1]\} \tag{2}$$

For example, if Q is the set of the states of a fuzzy automaton $\widetilde{F}$ which has three states, and possible membership values (mv) which may be attributed to these states are $\{0.1, 0.3, 0.4, 0.6, 0.7, 0.95\}$, then:

$$\textit{Domain of } \mu_Q = Q = \{q_1,\ q_2,\ q_3\}$$

$$\textit{Range of } \mu_Q = \{0.1, 0.3, 0.4, 0.6, 0.7, 0.95\}$$

Then, at different times $t_1, t_2, \ldots$ as $\widetilde{F}$ is operating, states q_1, q_2, and q_3 may take different mv's. e.g. $\mu_Q^{t_1}(q_1) = 0.3,\ \mu_Q^{t_1}(q_2) = 0.7,\ \mu_Q^{t_2}(q_3) = 0.6$, etc.

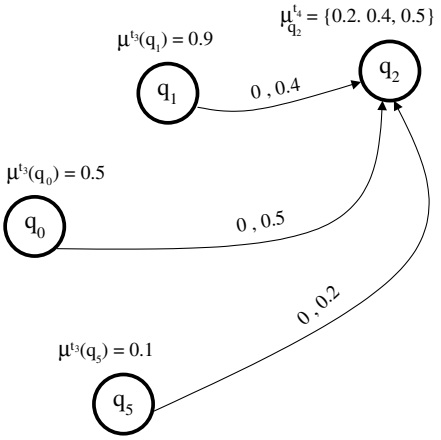

Fig. 1. Multi-membership of state q_2 at time t_4. See Example 1.

However, as will be seen, one of the interesting characteristics of fuzzy automata is the possibility of overlapping (simultaneous) transitions to the same state upon the same input symbol, but from different current states (see Fig. 1). This will lead to what we have called *multi-membership* (a next state is being forced to take several different membership values at the same time). To refer to multi-membership states we use a new notation μ_{q_i} with or without t.

To summarize, we state the following two conventions:

Convention 1. $\mu^t(q_m)$ refers to the single mv of state q_m at time t.

Whenever, time is unambiguous, we omit t and write simply: $\mu\ (q_m)$. For example, from Fig. 1 we can write: $\mu^{t_3}(q_1) = 0.9$, or if it is understood, $\mu(q_1) = 0.9$.

Convention 2. $\mu^t_{q_m}$ refers to the set of mv's associated with the multi-membership state q_m at time t.

Again, in case of no confusion, we may write: μ_{q_m}, e.g. in Fig. 1, we have: $\mu^{t_4}_{q_2} = \{0.2, 0.4, 0.5\}$ or $\mu_{q_2} = \{0.2, 0.4, 0.5\}$ as it is unambiguous.

In Section 3.2, we will show how $\mu^t_{q_m}$ is resolved to $\mu^t(q_m)$. See multi-membership resolution algorithm.

2.2. Insufficiency of the fuzzy automata literature

We believe that the current literature and background for fuzzy automata is not established well enough to characterize the operation of the automaton and thus fulfill the implementational requirements in a well-defined manner. We exemplify this shortage with the definition of a FFA. The following definition is generally accepted as a formal definition for FFA [33,47,46].

Definition 2. A Fuzzy Finite-state Automaton (FFA) $\widetilde{F}$ is a 6-tuple denoted as: $\widetilde{F} = (Q, \Sigma, \delta, R, Z, \omega)$, where:

Q is a finite set of states, $Q = \{q_1, q_2, \ldots, q_n\}$.
Σ is a finite set of input symbols, $\Sigma = \{a_1, a_2, \ldots, a_m\}$.
$R \in Q$ is the (possibly fuzzy) start state of $\widetilde{F}$.
Z is a finite set of output symbols, $Z = \{b_1, b_2, \ldots, b_k\}$.
$\delta: Q \times \Sigma \times Q \to (0, 1]$ is the fuzzy transition function which is used to map a state (current state) into another state (next state) upon an input symbol, attributing a value in the fuzzy interval $(0, 1]$ to the next state.
$\omega: Q \to Z$ is the output function which is used to map a (fuzzy) state to the output set. [2]

As can be seen, associated with each fuzzy transition, there is a membership value (mv) in $(0, 1]$ interval. We call this membership value the *weight* of the transition. The transition from state q_i (current state) to state q_j (next state) upon input a_k is denoted as $\delta(q_i, a_k, q_j)$. [3] Hereafter, we use this notation to refer both to a transition and its weight. Whenever $\delta(q_i, a_k, q_j)$ is used as a value, it refers to the weight of the transition. Otherwise, it specifies the transition itself.

There are two important problems which should be clarified in the definition of FFA. One is the assignment of membership values to the next states and the other is how output mapping is performed:

(1) *State Membership assignment:* There are two issues within state membership assignment, which need more elaboration. The first one is how to assign a mv to a next state upon the completion of a transition. Secondly, how should we deal with the cases where a state is forced to take several membership values simultaneously via overlapping transitions?

[2] Omlin et al. use the same definition both in [47,46]. While the use of an output map is somehow justifiable in [47], as it talks about the deterministic representation of FFA in recurrent neural networks, the need for that is not clear in [46], and it seems that ω (output map) is just included to keep consistency with the general definition of an automaton. On the other hand, Mordeson and Malik present several definitions of FFA in their book [33]. They all contain, Q, Σ, δ, and Z, but only in one of them is ω (output map) considered. Also, in their approach, start state is not of significance, while we do believe that a start state, specifically a fuzzy start state can affect the operation of a FFA significantly and produces quite different and more reasonable results.

[3] It should be noted that in [47,46] the transition mapping (δ) is represented as a relation, i.e. $\delta: Q \times \Sigma \times (0, 1] \to Q$. Thus, a transition from state q_i to state q_j upon input a_k with the transition weight W_{ijk} is denoted by: $\delta(q_i, a_k, W_{ijk}) = q_j$ [47,46]. But, since one of our main purposes is to develop a clear methodology for assigning membership values to states based on input strings, we follow [33] and represent the transition mapping as a function, i.e. δ: $Q \times \Sigma \times Q \to [0, 1]$ to put emphasis on the attribution of membership value. Thus the above transition will be denoted as $\delta\ (q_i, a_k, q_j) = W_{ijk}$. We use the term "function" here to imply that every value in the domain is related to a unique value in the range. This constraint makes mv assignment consistent with other fuzzy set formulations. It also allows us to simplify our definitions and subsequent derivations without loss of generality.

(2) *Output mapping:* The most important questions concerning the generation of output, are: What is the significance of the output map in fuzzy automata? Can we ignore that and still be able to deal with applications which do need different output labels?

The two issues of membership assignment can be seen in the following example.

Example 1. Consider the Fuzzy Finite-state Automaton (FFA) $\widetilde{F}$ as:

$$\widetilde{F} = (\Sigma, Q, R, Z, \delta, \omega)$$

where:

$\Sigma = \{0, 1\}$: set of input symbols.
$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$: set of states.
$R = \{q_0\}$: set of initial states. We assume the mv of q_0 is 1 at the beginning (i.e $\mu^{t_0}(q_0) = 1.0$).
$Z = \{accept\}$: set of output labels.
$\delta : Q \times \Sigma \times Q \to (0, 1]$: transition function.
$\omega : Q \to Z$: output map.

Here, there is a single output label. Although it can be called anything, *accept* seems to be more reasonable and consistent with the traditional automata terminology.

The automaton $\widetilde{F}$ and its transition table are shown in Fig. 2 and Table 1, respectively.

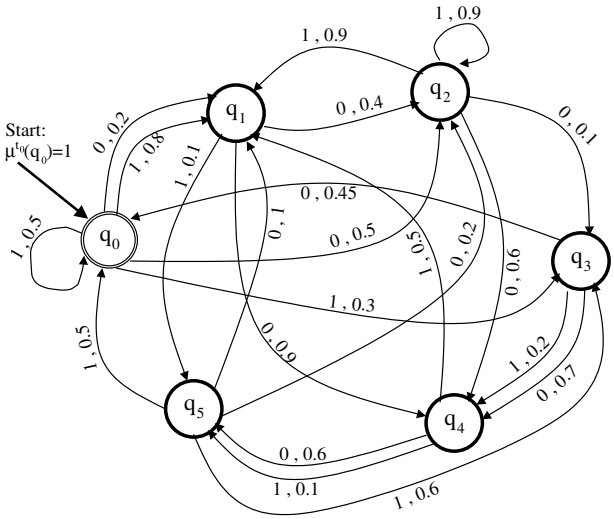

Fig. 2. **Illustration of FFA of Example 1.** A transition from the current state to the next state is shown by an arrow and the numbers on each arrow show the input symbol and the weight of transition respectively, separated by a comma ",". Thus, "1,0.2" shows that the transition occurs upon input "1" and the transition weight is "0.2".

Table 1
Transition table of FFA in Example 1

		next state					
		q_0	q_1	q_2	q_3	q_4	q_5
current state	q_0	1 , 0.5	0 , 0.2 1 , 0.8	0 , 0.5	1 , 0.3		
	q_1			0 , 0.4		0, 0.9	1 , 0.1
	q_2		1 , 0.9	1 , 0.9	0 , 0.1	0 , 0.6	
	q_3	0 , 0.45				0 , 0.7 1 , 0.2	
	q_4		1 , 0.5				0 , 0.6 1 , 0.1
	q_5	1 , 0.5	0 , 1.0	0, 0.2	1 , 0.6		

Consider now the operation of $\widetilde{F}$ upon input string "0110". The transitions are shown in Table 2.

To assign mv's to next states in this example, we use a currently accepted method which takes the weight of the transition [46,33]. We call this method *transition-based membership*. The disadvantage of transition-based membership and a more general method for assigning mv's to the states will be discussed in Section 3.1.

Using transition-based membership, the mv of a next state upon each input is the weight associated with the corresponding transition. Assuming that $\widetilde{F}$ started operation at time t_0, up to the third input symbol (i.e. "011") all (next) states take a unique mv, as there is a single transition to each state upon each symbol. e.g. $\mu^{t_1}_{q_1} = \{0.2\}$, $\mu^{t_1}_{q_2} = \{0.5\}$, $\mu^{t_2}_{q_1} = \{0.9\}$, $\mu^{t_3}_{q_3} = \{0.6\}$, etc. Since, associated with each state, there is a single mv, that can be assigned as the mv of that state at the specified time. Thus, in the above cases for example, the mv of q_1 at time t_1 is 0.2, i.e. $\mu^{t_1}(q_1) = 0.2$. Similarly, $\mu^{t_1}(q_2) = 0.5$, $\mu^{t_2}(q_1) = 0.9$, $\mu^{t_3}(q_3) = 0.6$. But, after the fourth symbol ("0") we have several states with overlapping transitions to them. For example q_1 is to take the mv 0.2 by the transition $\delta(q_0,0,q_1) = 0.2$, while at the same time, the transition $\delta(q_5,0,q_1) = 1.0$ forces it to 1.0, i.e. $\mu^{t_4}_{q_1} = \{0.2, 1.0\}$. Also q_2 at time t_4 has three associated mv's simultaneously, $\mu^{t_4}_{q_2} = \{0.2, 0.4, 0.5\}$ (see Fig. 1). Similarly, q_4 has three simultaneous associated transitions, $\mu^{t_4}_{q_4} = \{0.6, 0.7, 0.9\}$ In many situations, we may need to assign a single mv to each state, at any time (e.g. we may need $\mu^{t_4}(q_1)$, $\mu^{t_4}(q_2)$, and $\mu^{t_4}(q_4)$ in these examples). What should the mv of these states be at the specified times?

Table 2
Transitions of FFA in Example 1 upon input string "0110"

time	t_1	t_2	t_3	t_4
input	0	1	1	0
t	$\delta(q_0, 0, q_1) = 0.2$	$\delta(q_1, 1, q_5) = 0.1$	$\delta(q_5, 1, q_0) = 0.5$	$\delta(q_0, 0, q_1) = 0.2$
r				$\delta(q_0, 0, q_2) = 0.5$
a			$\delta(q_5, 1, q_3) = 0.6$	$\delta(q_3, 0, q_0) = 0.45$
n				$\delta(q_3, 0, q_4) = 0.7$
s	$\delta(q_0, 0, q_2) = 0.5$	$\delta(q_2, 1, q_1) = 0.9$	$\delta(q_1, 1, q_5) = 0.1$	$\delta(q_5, 0, q_1) = 1.0$
i				$\delta(q_5, 0, q_2) = 0.2$
t		$\delta(q_2, 1, q_2) = 0.9$	$\delta(q_2, 1, q_1) = 0.9$	$\delta(q_1, 0, q_2) = 0.4$
i				$\delta(q_1, 0, q_4) = 0.9$
o			$\delta(q_2, 1, q_2) = 0.9$	$\delta(q_2, 0, q_3) = 0.1$
n				$\delta(q_2, 0, q_4) = 0.6$

We call this problem *multi-membership*, which is one of the issues that has not been well-addressed in the literature and needs more elaboration. This will be the subject of Section 3.2.

2.3. Establishing a terminology for FFA

The terminology which is currently used for fuzzy automata, is essentially the same as the one used for other conventional automata such as DFA and PA. For example, it is not quite reasonable to use terms such as *current state* and *next state* for a FFA, as they are fuzzy and not unique. It is more sensible to talk about the *set of current fuzzy states*, the *set of next fuzzy states*, etc.

In the following, we will define some terminology to make the boundary between fuzzy automata and other types of automata more distinct, while at the same time the generality of fuzzy automata will be seen.

As we know, in the transition $\delta(q_m, a_k, q_j)$, q_m is traditionally known as the *current state* and q_j as the *next state*. But, for the fuzzy automata, we suggest to use the terms *successors* and *predecessors* for the reasons which will become clear as we proceed.

Convention 3. Unless otherwise specified, by successors and predecessors of a state q_m we mean states which follow q_m or are followed by q_m, respectively, within the context of a single input symbol.

Usually we need to refer to the set of all transitions in a fuzzy automaton (the set which specifies the domain of the function δ).

Convention 4. The set of all transitions of a fuzzy automaton $\widetilde{F}$, is denoted as $\Delta_{\widetilde{F}}$.

However, whenever it is understood we eliminate the subscript, and write simply Δ.

To show which states will be activated upon an input symbol from a specific state q_m, we define the set of successors of q_m as follows.

Definition 3 (*Successor set*). The successor set of a state q_m on input symbol a_k denoted as $Q_{\text{succ}}(q_m, a_k)$, is the set of all states q_j which will be reached via transitions $\delta(q_m, a_k, q_j)$.

$$Q_{succ}(q_m, a_k) = \{q_j \mid \delta(q_m, a_k, q_j) \in \Delta\} \tag{3}$$

It is sometimes desirable to define the successor set of a state subject to a threshold.

Definition 4 (*Threshold successor set*). The τ_1/τ_2 successor set of a state q_m on input symbol a_k denoted as $Q_{succ}(q_m, a_k, \tau_1/\tau_2)$, is the set of all successors of q_m like q_j such that $\tau_1 \leqslant \delta\ (q_m, a_k, q_j) \leqslant \tau_2$.

$$Q_{succ}(q_m, a_k, \tau_1/\tau_2) = \{q_j | \delta(q_m, a_k, q_j) \in \Delta \wedge \tau_1 \leqslant \delta(q_m, a_k, q_j) \leqslant \tau_2\} \tag{4}$$

Similarly, we can define the predecessor set ($Q_{\text{pred}}(q_m, a_k)$), and threshold predecessor set ($Q_{\text{pred}}(q_m, a_k, \tau_1/\tau_2)$) of a state as follows:

$$Q_{pred}(q_m, a_k) = \{q_j \mid \delta(q_j, a_k, q_m) \in \Delta\} \tag{5}$$

$$Q_{pred}(q_m, a_k, \tau_1/\tau_2) = \{q_j \mid \delta(q_j, a_k, q_m) \in \Delta \wedge \tau_1 \leqslant \delta(q_j, a_k, q_m) \leqslant \tau_2\} \tag{6}$$

In other words, q_m is the (threshold) successor of its (threshold) predecessors

$$Q_{pred}(q_m, a_k) = \{q_j \mid q_m \in Q_{succ}(q_j, a_k)\} \tag{7}$$

$$Q_{pred}(q_m, a_k, \tau_1/\tau_2) = \{q_j \mid q_m \in Q_{succ}(q_j, a_k, \tau_1/\tau_2)\} \tag{8}$$

It is obvious that:

$$Q_{succ}(q_m, a_k, \tau_1/\tau_2) \subseteq Q_{succ}(q_m, a_k) \subseteq Q \tag{9}$$

$$Q_{pred}(q_m, a_k, \tau_1/\tau_2) \subseteq Q_{pred}(q_m, a_k) \subseteq Q \tag{10}$$

It is also noticeable that the $0/\tau$, $\tau/1$, and τ/τ successor/predecessor sets of state q_m give all successors/predecessors of q_m whose transition weights are less than or equal to τ, greater than or equal to τ, and exactly equal to τ, respectively. Successor sets are specified as follows. Predecessor sets are quite similar.

$$Q_{succ}(q_m, a_k, 0/\tau) = \{q_j | \delta(q_m, a_k, q_j) \in \Delta \ \wedge \ \delta(q_m, a_k, q_j) \leqslant \tau\} \tag{11}$$

$$Q_{succ}(q_m, a_k, \tau/1) = \{q_j | \delta(q_m, a_k, q_j) \in \Delta \ \wedge \ \delta(q_m, a_k, q_j) \geqslant \tau\} \tag{12}$$

$$Q_{succ}(q_m, a_k, \tau/\tau) = \{q_j | \delta(q_m, a_k, q_j) \in \Delta \ \wedge \ \delta(q_m, a_k, q_j) = \tau\} \tag{13}$$

Definition 5 (*Active state set*). Knowing that the entered input prior to time t has been a_k, active states at time t are those states to which there is at least one transition on the input symbol a_k. Then, the fuzzy set of all active states at t (ordered pairs of states and their mv's) is called *active state set at time t*, and is denoted as $Q_{act}(t)$.

For example, in Fig. 2, assuming that the entered input at time t_0 is "0", we have:

$$Q_{act}(t_1) = \{(q_1, \mu^{t_1}(q_1)),\ (q_2, \mu^{t_1}(q_2))\} = \{(q_1, 0.2),\ (q_2, 0.5)\}$$

While if the entered input at t_0 is "1", we will have:

$$\begin{aligned} Q_{act}(t_1) &= \{(q_0, \mu^{t_1}(q_0)),\ (q_1, \mu^{t_1}(q_1)),\ (q_3, \mu^{t_1}(q_3))\} \\ &= \{(q_0, 0.5),\ (q_1, 0.8),\ (q_3, 0.3)\} \end{aligned}$$

Convention 5. Since $Q_{act}(t)$ is a fuzzy set (is a function), to show that a state q_i belongs to $Q_{act}(t)$, we should write: $q_i \in Domain(Q_{act}(t))$ (alternatively, $(q_i, \mu^t(q_i)) \in Q_{act}(t)$). Hereafter, we simply denote it as: $q_i \in Q_{act}(t)$, if no confusion occurs.

Obviously $Q_{act}(t) \subseteq \widetilde{P}(Q)$ for all t. $Q_{act}(t)$ can be defined recursively in time as:

$$Q_{act}(t) = \{(q_m, \mu^t(q_m)) \mid \exists (q_i \in Q_{act}(t-1), a_k \in \Sigma) \wedge q_m \in Q_{succ}(q_i, a_k)\} \tag{14}$$

Note that $Q_{act}(t_0) = \widetilde{R}$, i.e. at time t_0 the active state set, is the set of initial fuzzy states. In Fig. 2, for example: $Q_{act}(t_0) = \{(q_0, \mu^{t_0}(q_0)\} = \{(q_0, 1.0)\}$

In this paper, we refer abundantly to the classes of FFA with and without final states. To distinguish between them, we use two different notations.

Convention 6. FFA_{fin} refers to the class of FFA with some final states and FFA_{nofin} refers to the class of FFA without final states.

2.4. Conventional spectrum of automata

In our work, we sometimes refer to the conventional spectrum of automata. As mentioned previously, by conventional spectrum we mean: Deterministic Finite-state Automata (DFA), Non-deterministic Finite-state Automata (NFA), Probabilistic (stochastic) Automata (PA), and Fuzzy-Finite-state Automata (FFA). We have summarized and compared the conventional spectrum in Table 3 which will be referred hereafter from time to time.

We just note some points which are of our concern to pave the way for our contributions. For more details see [48,8,49,50,5,51].

Table 3
Comparison of conventional automata

	DFA	NFA	PA	FFA
R	Single start state(q_0)	Set of start states	Single probabilistic start state	Set of fuzzy start states
Z	Accept/reject	Accept/reject	Accept/reject	Acceptance according to mv's
ω	Set of final (acceptance) states (F) $\omega(q) = \begin{cases} reject & \text{if } q \in F \\ accept & \text{if } q \notin F \end{cases}$	Set of final (acceptance) states (F) $\omega(q) = \begin{cases} reject & \text{if } q \in F \\ accept & \text{if } q \notin F \end{cases}$	Probabilistic final (acceptance) states	Possible fuzzy final states (see Section 3)
δ	$Q \times \Sigma \to Q$	$Q \times (\Sigma \cup \{\epsilon\}) \to 2^Q$	$Q \times \Sigma \times (0,1] \to Q$	$Q \times \Sigma \times Q \to (0,1]$
$Range(\mu_Q)$	$\{0,1\}$	$\{0,1\}$	$\{0,1\}$	$(0,1]$
$\lvert Q_{succ}\rvert$	1	$\geqslant 0$	1	$\geqslant 1$
$\lvert Q_{pred}\rvert$	1	$\geqslant 0$	1	$\geqslant 1$

An automaton A is considered as a 6-tuple machine as in Definition 2, i.e. $A = (\Sigma, Q, R, Z, \delta, \omega)$. $q \in Q$ refers to a general state and μ_Q is the function (fuzzy set) which mapps q to a mv (see Definition 1).

(1) NFA have the characteristic of ϵ-transition. An NFA can transit from one state to another on empty string (ϵ). Although this property does not make sense practically, it is very useful in developing the theory of formal languages and theorem proving. We will use this theoretically constructive property in a different sense for FFA.
(2) In NFA, like FFA overlapping of transitions is possible. However, since δ is applied non-deterministically but not probabilistically, and due to ϵ-transition, there is no way to define deterministic behaviour of a NFA. However, had it been possible to do that, there would have been no vagueness and fuzziness in NFA operation as all active states, successors, and predecessors would have had an implied mv of 1. This means that the sheer overlapping of transitions does not imply fuzziness. Moreover, any NFA M can be unrolled (unfolded) to a DFA M' (which can have considerably more states and is said to be equivalent to M in terms of accepting language, i.e. $L(M) = L(M')$) where at any time there is a unique active state (with an implied mv of 1) with a unique successor and a unique predecessor [49]. But, generally, there is no way to simulate the operation of a fuzzy automaton by a DFA, or any other type of automata in the conventional spectrum of automata (or even any combination of them). However, it may be possible to derive DFA which are equivalent to some specific and narrowed cases of fuzzy grammars [52]. But, this can not be generalized to a general class of fuzzy automata. See Definition 8.
(3) All transitions of a DFA and NFA have an implied weight of 1, while the weights of transitions in a PA and FFA belong to $(0, 1]$. However, in all types of conventional automata, a zero-weight transition means no transition, while in our approach to fuzzy automata, a zero-weight transition does not necessarily imply no transition. That is why we will use $[0, 1]$ as the fuzzy interval. See Section 3.1.
(4) Although PA and FFA seem to be similar, they have two major differences:
    (a) Operationally in a PA, at any specific time, there is a single active state which has exactly one predecessor and one successor (although the successor is not predictable, if more than one potential successor exist). The weights are in fact probabilities of transition, which cause δ to be applied randomly. When the PA is put into operation, there is no vagueness in the current state, next state, and the extent to which they will be activated. At any time (upon each input symbol), one and exactly one state will be activated with an implied membership value of 1.
    (b) In a PA, the sum of the weights for the transitions (from the current state) corresponding to a specific input symbol should be one, whilst there is no such requirement for a FFA.
(5) In so far as we are considering binary output mode (reject/accept or $\{0, 1\}$), in all types of automata, input symbols (strings) are either accepted or rejected, except in FFA where the acceptance or rejection is a matter of graded membership. Hence, strings will be accepted (rejected) to some extent according to their mv's. However, some people talk about full membership and non-membership in a FFA, i.e. some inputs may be completely accepted (full membership, $\mu = 1$) or completely rejected (non-membership, $\mu = 0$). This is to our mind a con-

troversial issue which will be clarified later. In fuzzy automata, talking about full membership and non-membership does not seem to be very sensible in all but a degenerate case.

(6) Compared to other automata in the conventional spectrum, fuzzy automata are more general. We have included the function μ_Q (which maps membership values to states) in Table 3 to emphasize this point. Yet, FFA are not general enough to represent completely other automata in the conventional spectrum. That is why, we will present a new and general definition for fuzzy automata in Section 3.4 (Definition 8), which not only encompasses all types of automata (including conventional fuzzy automata as in Definition 2), but also several other computational paradigms. See also [39] for more details.

3. Establishing stronger definitions and methodologies for fuzzy automata

In the previous section, we addressed some shortages of the current literature of FFA. Now we introduce some solutions to deal with these shortages. We will first elaborate the two essential problems of membership assignment and output mapping. Then based on this elaboration, we will present a new and more general definition for fuzzy automata and develop a methodology to analyze the continuous operation of FFA.

3.1. Membership assignment

The way mv's are assigned to active states, requires further analysis. Currently, there is a generally approved approach for assigning mv to a next state (whether it is final or not), where we just use the weight of the transition, and ignore the membership value of the current state [46]. Thus, the weight of the transition will be considered as the mv of the next state. We called this approach *transition-based membership*. See Example 1.

Although the transition-based membership can work well for fuzzy automata realized based on certain types of fuzzy grammars, it has some disadvantages which make it unsuitable for some applications. To see the consequence of transition-based membership, consider the following example:

Example 2. Suppose in a specific FFA the membership value of the state q_1 (current state) at time t is 0.01 and the weight of the transition upon input symbol a to the next state (q_2) is 1.0. The FFA is partially shown in Fig. 3.

Using transition-based membership and assuming that the input symbol upon time t is a, we have:

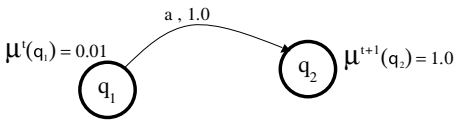

Fig. 3. A full activation caused by a weak activation.

$$\delta(q_1, a, q_2) = 1.0 \Rightarrow \mu^{t+1}(q_2) = 1.0$$

This means that a state which is active to an extent of 0.01 ($\mu^t(q_1) = 0.01$) has caused its successor to be fully activated ($\mu^{t+1}(q_2) = 1.0$). Obviously, such an extension without considering the level of activation of the predecessor is not always reasonable. Even, if in a specific application, the mv of q_2 becomes 1.0 in such a situation, it should be assured that mv assignment has been done considering the level of activation of the predecessor (for example the maximum of the predecessor mv and the weight of the transition may have been assigned to the successor). The goal of this simple example however, is to show the insufficiency of transition-based membership as a general membership assignment process.

3.1.1. Augmented transition function: A general method to assign membership values

To establish a general method to assign mv's to next states, we generalize the definition of the transition function in fuzzy automata (Definition 2). This generalization enables it to incorporate both the level of activation of the current state and the weight of the transition.

In Definition 2, δ was defined as: $\delta : Q \times \Sigma \times Q \to (0, 1]$ and the weight of the transition from state q_i to q_j upon input a_k was denoted as $\delta(q_i, a_k, q_j)$.

Now, we define a new transition function $\widetilde{\delta}$, which we call *augmented transition function*. $\widetilde{\delta}$ is represented as:

$$\widetilde{\delta} : (Q \times [0, 1]) \times \Sigma \times Q \overset{F_1(\mu,\delta)}{\to} [0, 1] \tag{15}$$

$\widetilde{\delta}$ maps the active state (reached from its predecessor), to the fuzzy interval $[0, 1]$ via function $F_1(\mu, \delta)$. We call F_1 the *membership assignment function*, which is defined as:

Definition 6 (*Membership assignment function*). In an FFA, membership assignment function is a mapping function which is applied via augmented transition function $\widetilde{\delta}$ to assign mv's to the active states.

$$F_1 : [0, 1] \times [0, 1] \to [0, 1] \tag{16}$$

Function $F_1(\mu, \delta)$ as is seen, is motivated by two parameters:

- μ: the mv of a predecessor;
- δ: the weight of a transition.

In this new definition, the process that takes place upon the transition from state q_i to q_j on input a_k is represented as:

$$\mu^{t+1}(q_j) = \widetilde{\delta}((q_i, \mu^t(q_i)), a_k, q_j) = F_1(\mu^t(q_i), \delta(q_i, a_k, q_j)) \tag{17}$$

which means that the mv of the state q_j at time $t + 1$ is computed by function F_1 using both the mv of q_i at time t and the weight of the transition.

There are many options which can be used for the function $F_1(\mu, \delta)$. The best option however, depends on the application at hand. It can be for example Max, Min,

Mean or any other applicable mathematical function. We can also incorporate time into its evaluation. Its exact form depends on the application. However, it should satisfy the following axioms:

Axiom 1. $0 \leqslant F_1(\mu, \delta) \leqslant 1$.

Axiom 2. $F_1(0,0) = 0$ and $F_1(1,1) = 1$.

Axiom 2 guarantees the boundary conditions. Some examples of $F_1(\mu, \delta)$ are:
$F_1(\mu, \delta) = Mean(\mu, \delta) = \frac{\mu+\delta}{2}$ (arithmetic mean)
$F_1(\mu, \delta) = GMean(\mu, \delta) = \sqrt{\mu \cdot \delta}$ (geometric mean)
$F_1(\mu, \delta) = \begin{cases} \mathrm{Max}(\mu, \delta) & \text{if } t < t_i \\ \mathrm{Min}(\mu, \delta) & \text{if } t \geqslant t_i \end{cases}$
$F_1(\mu, \delta) = \mathrm{Min}[1, (\mu^\omega + \delta^\omega)^{1/\omega}],\ \omega > 0$ (Yager class of t-conorms [53])

It should be noted that transition-based membership can be considered as a special case of the augmented transition function, where we assume that $F_1(\mu, \delta) = \delta$. In other words, in this case, the level of activation of the predecessors (μ) has no significance in attribution of mv to active states. Example 1 illustrated transition-based membership.

This new method of membership assignment has several advantages and consequences

(1) The first advantage is its generality and flexibility. It not only entails all other methods reported in the literature, but also is more easily adapted to the requirements of specific applications.
(2) A more important advantage is that, together with a multi-membership resolution strategy discussed next, it paves the way to develop a general structured method to assign mv's to input strings (rather than a single input symbol) and to analyze the continuous operation of fuzzy automata as will be seen in Section 3.5.
(3) A very interesting consequence of this approach is the distinction between zero-weight transition and no transition. A zero-weight transition is possible and can cause a successor to get activated via the activation of its predecessor, provided that the function F_1 defined for the fuzzy automaton permits such a contribution. However, the practical usefulness of this characteristic remains to be seen.

The augmented transition function ($\widetilde{\delta}$) gives essentially a kind of memory to FFA to remember the previous state when it reaches a new state. The mv of the predecessor will be memorized and is used by the augmented transition function ($\widetilde{\delta}$) upon reaching a successor. This method can even be extended further to remember not only the membership value of the current active state, but also a chain of the previous states (and transitions) to give more ability to FFA. However, the usefulness of such an extension is not quite obvious yet and needs more investigation. For now, we restrict ourselves to one level of memorization.

3.2. Multi-membership resolution

The second issue to be addressed, is that of *multi-membership*, which is again a problem arising due to the fuzzy nature of FFA (see Example 1).

Very little can be found in the literature which address the problem of multi-membership and how it should be treated. One of the approaches for tackling multi-membership is the one taken by Omlin et al. [46]. They call multi-membership "ambiguity", and devise a method to remove ambiguities. They develop an algorithm which creates a new state for each overlapping transition, whenever the transition weights conflict, i.e. are not equal. Hence, there will be no multi-memberships after this algorithm is applied to a FFA. In [46], Omlin et al. are addressing the direct representation of FFA in Recurrent Neural Networks (RNNs). What they do (which they call ambiguity removal) is in fact a way to make their representational scheme possible. A simple example of ambiguity removal is illustrated in Fig. 4.

The disadvantage of the ambiguity removal is twofold. First, it increases the number of states significantly. Roughly speaking, it creates one new state per overlapping transitions to the same state. Second, the original FFA will change which may be unacceptable or infavorable for many applications.

In essence, however, the term ambiguity to our mind is not an appropriate name for multi-membership. Multi-membership is something inherent to the FFA and happens due to its fuzzy nature. It shows up under almost any situation. Therefore, it should be resolved in an appropriate way, based on the system under consideration to fulfill its requirements. The following reasons clarify the necessity of such a resolution:

(1) If the multi-membership active state is final, the necessity of a single mv is obvious, as a final state is usually used to produce a crisp output (after defuzzification) for the system under consideration.
(2) Even, if the multi-membership active state is not final, in some applications, we may need to assign a single mv to some intermediate (non-final) states during the normal operation of a FFA. Such a need may arise for example when we have a continuous flow of input symbols and in the meantime we need to take some actions based on the mv's of some active states, which necessitates a single mv.
(3) The mv's of successors can be computed for each mv of the current state. But, this will lead to unnecessary blow up which makes tracing the continuous operation of FFA very difficult if not impossible. Moreover, our observations show that for practical applications we do not need this way of handling multi-membership. It suffices to resolve each multi-membership active state and attribute a single mv to that state. This unified mv can then be used as the μ parameter in $F_1(\mu, \delta)$ function to compute the mv's of successors of that state.

Therefore, at any stage to compute the mv of the next states or to make a decision based on the mv of the current active state(s), we need a single value to be used as the

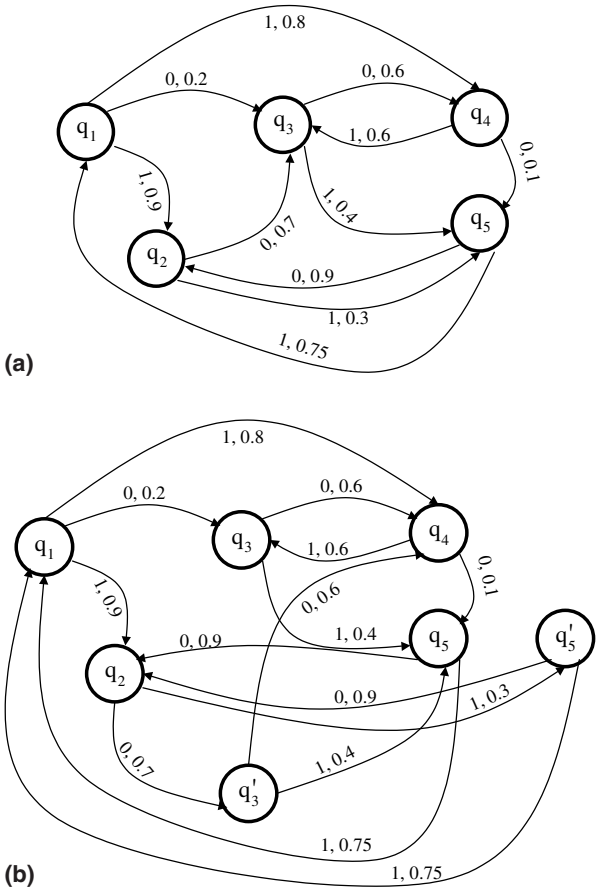

Fig. 4. *Ambiguity removal.* (a) FFA before ambiguity removal. (b) FFA after ambiguity removal. Note how multi-memberships of states q_3 (on input "0") and q_5 (on input "1") are removed by creating new states q'_3 and q'_5, respectively. Although after ambiguity removal states q_1, q_2, q_4, q_5 have overlapping transitions, they are not multi-membership, as there is no conflict and the weights of overlapping transitions are equal. For more details see [46].

level of activation of each active state. If there is only one mv associated with such a state, we have no problem. But, if the state is to take several mv's simultaneously, how can we assign a single mv to it? [4]

In other words, to tackle the problem of multi-membership, we need to establish a process which we call *multi-membership resolution*. This is done by another function F_2. We call F_2 *multi-membership resolution function*, and define it as:

[4] Note that the ambiguity removal presented in [46] does not provide any answer for this question. In fact, from an applicational point of view, it makes the problem more ambiguous, should both the predecessor and the transition weight contribute to the mv assignment of the successor via F_1.

Definition 7 (*Multi-membership resolution function*). In an FFA, the multi-membership resolution function, is a function which specifies the strategy, that resolves the multi-membership active states and assigns a single mv to them. It is represented as:

$$F_2 : [0,1]^* \to [0,1] \tag{18}$$

Then, the combination of the operations of functions F_1 and F_2 on a multi-membership state q_m will lead to the multi-membership resolution algorithm.

Algorithm 1 (*Multi-membership resolution*). If there are several simultaneous transitions to the active state q_m at time $t+1$, the following algorithm will assign a unified mv to that:

(1) Each transition weight $\delta(q_i, a_k, q_m)$ together with the mv of the corresponding predecessor q_i, will be processed by the membership assignment function F_1 (via augmented transition function δ), and will produce a mv. Call this v_i.

$$v_i = \widetilde{\delta}((q_i, \mu^t(q_i)), a_k, q_m) = F_1(\mu^t(q_i), \delta(q_i, a_k, q_m)) \tag{19}$$

(2) These mv's (v_i's) are not necessarily equal. Hence, they will be processed by another function F_2, called the multi-membership resolution function.
(3) The result produced by F_2 will be assigned as the instantaneous mv of the active state q_m.

$$\mu^{t+1}(q_m) = \overset{n}{\underset{i=1}{F_2}}[v_i] = \overset{n}{\underset{i=1}{F_2}}[F_1(\mu^t(q_i), \delta(q_i, a_k, q_m))] \tag{20}$$

where

- n: is the number of simultaneous transitions from states q_i's to state q_m prior to time $t+1$ and $q_i \in Q_{\text{pred}}(q_m, a_k)$, i.e. n is the cardinality of the set $\mu^{t+1}_{q_m}$.
- $\delta(q_i, a_k, q_m)$: is the weight of the transition from q_i to q_m upon input a_k.
- $\mu^t(q_i)$: is the membership value of q_i at time t (possibly resolved, i.e. unified).
- $\mu^{t+1}(q_m)$: is the final mv of q_m at time $t+1$.

Similar to F_1, there are many options applicable to F_2. The best strategy should be selected based on the application at hand. However, the following axioms are the minimum requirements to be satisfied by F_2:

Axiom 3. $0 \leqslant \overset{n}{\underset{i=1}{F_2}}(v_i) \leqslant 1$.

Axiom 4. $F_2(\phi) = 0$.

This axiom essentially, paves the way for the ϵ-transition to be incorporated into the operation of fuzzy automata.

Axiom 5. $\overset{n}{\underset{i=1}{F_2}}(v_i) = a$ if $\forall i(v_i = a)$.

Whenever, all predecessors of a multi-membership state produce the same mv, it is reasonable that the active state assumes this mv. This is the logic behind Axiom 5. An immediate corollary of this axiom is:

$$\overset{n}{\underset{i=1}{F_2}}(v_i) = 0 \text{ if } \forall i\ (v_i = 0) \quad \text{and} \quad \overset{n}{\underset{i=1}{F_2}}(v_i) = 1 \text{ if } \forall i (v_i = 1).$$

Another useful corollary of this axiom is:

$$\overset{1}{\underset{i=1}{F_2}}(v_i) = v_i$$

which enables F_2 to be considered as a general process operating on all active states, no matter if they are multi-membership or not. This generalization is specifically useful for representation and learning of general fuzzy automata [54].

We are currently modeling some applications using different FFA, and applying neural network training algorithms to them. For training purposes many different strategies can be selected for F_2. We mention maximum and mean strategies as examples. However, the best-fitted strategy for any application should be selected based on the requirements of that application.

- Maximum multi-membership resolution

$$\mu^{t+1}(q_m) = \underset{i=1 \text{ to } n}{\text{Max}}[\widetilde{\delta}((q_i, \mu^t(q_i)), a_k, q_m)] = \underset{i=1 \text{ to } n}{\text{Max}}[F_1(\mu^t(q_i), \delta(q_i, a_k, q_m))] \tag{21}$$

- Arithmetic mean multi-membership resolution

$$\mu^{t+1}(q_m) = \left[\sum_{i=1}^{n} \widetilde{\delta}((q_i, \mu^t(q_i)), a_k, q_m)\right] \Big/ n = \left[\sum_{i=1}^{n} F_1(\mu^t(q_i), \delta(q_i, a_k, q_m))\right] \Big/ n \tag{22}$$

where n is the number of simultaneous transitions from q_i's to q_m at time $t + 1$, and $q_i \in Q_{\text{pred}}(q_m, a_k)$.

3.3. Output mapping

Generally, in most applications and systems there is some kind of final output or decision. In fuzzy clustering (classification) for example, we may have several final states each of which specify a different cluster with a specific label and its own mv. Hence, we need to attribute output values to the states of FFA. There are also some other applications where the final decision (output) of the system is a kind of acceptance or rejection, even though the system performs itself fuzzily.

However, in classical DFA and NFA there is no explicit output beyond the concept of acceptance attributed to each state, while in Moore machines an output (label) selected from a predefined output alphabet, is mapped to each state. Then DFA and NFA can be considered as specific cases of Moore machines where the output label is either acceptance (final states) or rejection (non-final states) [49].

To keep consistency with the conventional spectrum, we consider the FFA with final states (FFA_{fin}) as a special case of *Fuzzy Moore Finite-state Automata (FMFA)*, where there is a binary output label set ($Z = \{accept, reject\}$). Still, the FFA with no final state (FFA_{nofin}) can be considered as another special case of FMFA, where there is a single output label ($Z = \{accept\}$ say). The most important point however, is that in all these cases, the output activation is a matter of the extent to which these states are activated. This is the point which differentiates FFA from their classical counterparts (DFA, NFA, PA, Moore machines, Mealy machines, etc.).

3.4. Computational generality of fuzzy automata

During the past few years, several researchers studied the equivalence and isomorphism of fuzzy automata with other types of automata [55–57]. However, we do believe that the role of fuzzy automata goes beyond equivalence. In fact, they can represent not only other types of automata, but also several other computational paradigms [39]. To make the computational generality of fuzzy automata and its generalization capability more systematic and application-friendly, we need a more general definition. In the sequel, with the incorporation of output mapping in the sense discussed above and also the incorporation of F_1 and F_2, we present a new definition for FFA which is much more general compared to the current ones.

3.4.1. A new definition for fuzzy automata

Definition 8 (*General fuzzy automaton*). A General Fuzzy Automaton (GFA) $\widetilde{F}$ is an 8-tuple machine denoted as $\widetilde{F} = (Q, \Sigma, \widetilde{R}, Z, \omega, \widetilde{\delta}, F_1, F_2)$, where:
Q is a finite non-fuzzy set of states, $Q = \{q_1, q_2, \ldots, q_n\}$.
Σ is a finite non-fuzzy set of input symbols, $\Sigma = \{a_1, a_2, \ldots, a_m\}$.
$\widetilde{R}$ is the set of fuzzy start states. $\widetilde{R} \subseteq \widetilde{P}(Q)$, see Definition 1.
Z is a finite non-fuzzy set of output labels (symbols), $Z = \{b_1, b_2, \ldots, b_l\}$.
$\omega : Q \to Z$ is the non-fuzzy output function.
$F_1 : [0,1] \times [0,1] \to [0,1]$ is a mapping function, which is applied via $\widetilde{\delta}$ to assign mv's to the active states, thus called *membership assignment function*. See Definition 6.
$\widetilde{\delta} : (Q \times [0,1]) \times \Sigma \times Q \overset{F_1(\mu,\delta)}{\to} [0,1]$ is the augmented transition function.
$F_2 : [0,1]^* \to [0,1]$ is a multi-membership resolution strategy which resolves the multi-membership active states and assigns a single mv to them, thus called *multi-membership resolution function*. See Definition 7.

3.4.2. Specific cases of fuzzy automata

Definition 8 is very broad and general. It is actually the most general definition of an automaton (in discrete spaces), which entails other types of automata including conventional fuzzy automata, as special cases. On the other hand, the issues of out-

put mapping in fuzzy automata, needs more elaboration. In this section, we show how the generality of GFA contributes to this elaboration. For more details see [58]. Of course as will be seen, the generality of GFA is not restricted to fuzzy automata with outputs.

In deterministic realm, automata with outputs are categorized into Mealy machines [59] and Moore machines [60]. As we know however, using Moore and Mealy models is not restricted to DFA and the idea has been applied to other types of automata. For example, Bruce and Fu [48] developed a class of stochastic automata based on the Mealy model.

Following this line, we categorize GFA with outputs into Moore and Mealy models depending on whether the output labels are associated with states or transitions. This leads to Fuzzy Moore Finite-state Automata (FMFA) and Fuzzy MeaLy Finite-state Automata (FMLFA), which can be considered as specific cases of GFA.

Table 4 shows four specific classes of GFA in terms of output mapping. Regarding Table 4, some points should be noted.

(1) A fuzzy automaton without final states (FFA_{nofin}) can also be specified as follows:
Z contains a single output symbol, e.g. $Z = \{accept\}$.
$\omega: Q \to Z$, $\omega(q_i) = accept\ \forall q_i \in Q$. that implies: $Q_{fin} = Q$. However, due to simplicity, the definition mentioned in Table 4 is preferable.

(2) In FMLFA, it is possible to relate the output associated with the transitions only to the input symbol. This means that all transitions from state q_i upon input a_k will have the same output label, no matter what the successors will be. i.e. the output map is defined as: $\lambda: Q \times \Sigma \to Z$, and is represented as: $\lambda(q_i, a_k) = b_m$.
But, we take the general scenario and assume that every single transition may have its own output label, thus defining the output map as: $\lambda: Q \times \Sigma \times Q \to Z$, and representing it as: $\lambda(q_i, a_k, q_j) = b_m$.

Table 4
Output map for different classes of GFA

Class of GFA	Output set (Z[a])	Output map	Output assignment
FFA with final states (FFA_{fin})	$Z = \{accept, reject\}$	ω: $Q \to Z$	$\begin{cases} \omega(q_i) = accept \Rightarrow q_i \in Q_{fin} \\ \omega(q_i) = reject \Rightarrow q_i \notin Q_{fin} \end{cases}$ [b]
FFA without final states (FFA_{nofin})	$Z = \phi$	ω: not applicable	
Fuzzy Moore Finite-state Automata (FMFA)	$Z = \{b_1, b_2, \ldots, b_l\}$	ω: $Q \to Z$	at any time t, *if* $q_i \in Q_{act}(t) \Rightarrow \gamma\ (q_i) = b_j$
Fuzzy MeaLy Finite-state Automata (FMLFA)	$Z = \{b_1, b_2, \ldots, b_l\}$	λ: $Q \times \Sigma \times Q \to Z$	–[c]

[a] Z is a finite non-fuzzy set of output symbols.
[b] Q_{fin} is the set of final states ($Q_{fin} \subseteq Q$).
[c] At any time t, if $(q_i \in Q_{act}(t) \wedge$ subsequent input is $a_k \wedge q_j \in Q_{succ}(q_i, a_k)) \Rightarrow \lambda(q_i, a_k, q_j) = b_m$.

(3) Since all classes of Table 4 are specific cases of GFA, $\widetilde{\delta}$ is applied as in Definition 8, i.e.

$$\widetilde{\delta}((q_i, \mu(q_i)), a_k, q_j) = F_1(\mu(q_i), \delta(q_i, a_k, q_j))$$

(4) Although the output labels in FMFA and FMLFA are assigned by γ and λ, respectively, the level to which they will be activated, will be attributed by the successors through Algorithm 1 (combination of F_1 and F_2, see [58] for more details). However, it is noticeable that other options are possible for FMLFA, which make it more complicated, but at the same time give more capabilities to it. We can for example, assume that the level of activation of output labels will be assigned by the transitions separately. Then, the mv of the states and the mv of the output labels will be two different parameters, thus implying possibly two different characteristics of the application under consideration. We will talk more about the potentials of FMLFA in concluding remarks.

To avoid confusion from now on, we use $\widetilde{F}$ to denote a general fuzzy automaton (GFA), which may be a Moore or Mealy model or neither of them. But, the Moore and Mealy FFA, whenever used, will be explicitly denoted by $\widetilde{M}$ and $\widetilde{L}$, respectively.

3.5. Continuous operation of GFA

As mentioned before, we are mostly focussing on the operational issues relating to GFA. Hence, no matter if we are concerned with the existence of final state(s) or not and whether or not a single mv is needed for each active state (final or non-final) reached during the operation of a fuzzy automaton, we assume that in a GFA a single mv will be assigned to each active state, after applying the multi-membership resolution algorithm.

So far, we have talked about the transition and output mapping for a single input symbol, irrespective of the type of fuzzy automaton. In practice we need to extend these concepts to a string of input symbols. In the sequel, we extend the concepts of the membership set and mv to strings of input symbols. This extension

(1) Enables us to analyze the behaviour of GFA under successive input symbols.
(2) Opens the way to talk about the equivalence of two fuzzy automata and also devising algorithms to convert different fuzzy automata to each other.

We require more new terms to analyze the continuous operation of GFA.

Definition 9 (*Derivation*). A derivation of an input string x ($x \in \Sigma^*$) denoted as $der_i(x)$, is an ordered set of states which are passed successively upon entrance of each symbol of the string, starting from an initial state. i is an arbitrary index usually starting from 1.

Given that $x = a_1 a_2 \ldots a_k\ a_{k+1} \ldots a_m$, we have:

$$der_i(x) = \Big\{ q_{i_0} q_{i_1} \dots q_{i_k} q_{i_{k+1}} \dots q_{i_m} | q_{i_0} \in \widetilde{R} \wedge q_{i_0} \overset{\delta(q_{i_0}, a_1, q_{i_1})}{\longrightarrow} q_{i_1} \dots q_{i_k} \overset{\delta(q_{i_k}, a_{k+1}, q_{i_{k+1}})}{\longrightarrow} q_{i_{k+1}} \dots q_{i_{m-1}} \overset{\delta(q_{i_{m-1}}, a_m, q_{i_m})}{\longrightarrow} q_{i_m},\ 0 \leqslant k < m \Big\} \tag{23}$$

A string may have several derivations. The set of all derivations of string x is denoted as $D_{der}(x)$. Then, a threshold derivation of x is a member of $D_{der}(x)$ subject to a threshold, as follows:

Definition 10 (*Threshold derivation*). A τ_1/τ_2 derivation of an input string x denoted as $der_i(x, \tau_1/\tau_2)$, is defined as:

$$der_i(x, \tau_1/\tau_2) = \{ q_{i_0} q_{i_1} \dots q_{i_k} q_{i_{k+1}} \dots q_{i_m} \in D_{der}(x) | \tau_1 \leqslant \delta(q_{i_k}, a_k, q_{i_{k+1}}) \leqslant \tau_2,\ 0 \leqslant k < m \} \tag{24}$$

Similarly, the set of all threshold derivations of x is denoted as $D_{der}(x, \tau_1/\tau_2)$. Obviously: $D_{der}(x, \tau_1/\tau_2) \subseteq D_{der}(x)$.

Actually, we are interested in the active states of a fuzzy automaton upon entry of a string x. Without any loss of generality, we can use the same notation used for the active state set at a specific time ($Q_{act}(t)$), and denote the active state set of string x as $Q_{act}(x)$, since:

$$Q_{act}(x) \equiv Q_{act}(t_0 + |x|) \tag{25}$$

where t_0 is the starting time of operation of the GFA and $|x|$ is the length of x. Then, $Q_{act}(x)$ can be defined as:

Definition 11 (*Active state set of an input string*). The active state set of an input string x, is the fuzzy set of all active states, after string x has entered the GFA.

$$Q_{act}(x) = \{ (q_{i_m}, \mu^{t_0+|x|}(q_{i_m})) \mid q_{i_0} q_{i_1} \dots q_{i_k} q_{i_{k+1}} \dots q_{i_m} \in D_{der}(x) \} \tag{26}$$

Similarly, the *threshold active state set* of an input string is defined as:

$$Q_{act}(x, \tau_1/\tau_2) = \{ (q_{i_m}, \mu^{t_0+|x|}(q_{i_m})) \mid q_{i_0} q_{i_1} \dots q_{i_k} q_{i_{k+1}} \dots q_{i_m} \in D_{der}(x, \tau_1/\tau_2) \} \tag{27}$$

It is obvious that $Q_{act}(x, \tau_1/\tau_2) \subseteq Q_{act}(x) \subseteq \widetilde{P}(Q)$.

Here also, the $0/\tau$, $\tau/1$, and τ/τ thresholds will give the threshold derivations and active state sets, where all transitions are less than or equal to τ, greater than or equal to τ, and exactly equal to τ, respectively.

Example 3. For the fuzzy automaton of Example 1 (Fig. 2), assume that we use transition based membership ($F_1(\mu, \delta) = \delta$), and $F_2() = \text{Max}$. Then the following can be easily verified:

$$der_1(\text{“0110”}) = q_0 q_1 q_5 q_0 q_1, \quad der_2(\text{“0110”}) = q_0 q_1 q_5 q_0 q_2,$$
$$\dots \qquad \dots \qquad \dots \qquad \dots$$
$$der_9(\text{“0110”}) = q_0 q_2 q_2 q_2 q_3, \quad der_{10}(\text{“0110”}) = q_0 q_2 q_2 q_2 q_4$$

$$D_{der}(\text{“0110”}) = \{der_1(\text{“0110”}), \ldots, der_{10}(\text{“0110”})\}$$
$$= \{q_0q_1q_5q_0q_1, \ldots, q_0q_2q_2q_2q_4\}$$

$$der_1(\text{“0110”}, 0.5/0.9) = q_0q_2q_2q_2q_4$$

$$der_2(\text{“0110”}, 0.5/0.9) = q_0q_2q_2q_1q_4$$

$$D_{der}(\text{“0110”}, 0.5/0.9) = \{q_0q_2q_2q_2q_4, q_0q_2q_2q_1q_4\}$$

$$Q_{act}(\text{“0110”}) = \{(q_0, 0.45), (q_1, 1.0), (q_2, 0.5), (q_3, 0.1), (q_4, 0.9)\}$$

$$Q_{act}(\text{“011”}) = \{(q_0, 0.5), (q_1, 1.0), (q_2, 0.9), (q_3, 0.6), (q_5, 0.1)\}$$

$$Q_{act}(\text{“011”}, 0.1/0.5) = \{(q_1, 1.0), (q_2, 0.9)\}$$

$$Q_{act}(\text{“011”}, 0.2/0.5) = \phi$$

As was seen, for the class of FFA_{fin}, we defined Q_{fin}. To generalize this concept to the class of fuzzy automata with several output labels (specifically FMFA), we devise a new term, *State set of an output label*.

Definition 12 (*State set of an output label*). In the FMFA $\widetilde{M} = (Q, \Sigma, \widetilde{\delta}, R, Z, \gamma, F_1, F_2)$, the state set of the output label z_l denoted as Q_{z_l}, is the set of all states whose associated output is z_l.

$$Q_{z_l} = \{q_m | \gamma(q_m) = z_l\} \tag{28}$$

3.5.1. Acceptance and rejection in GFA

Usually, when we talk about acceptance and rejection in an automaton (as we do for DFA or NFA), it is implied that some states are final, and the existence of final states in turn implies a binary output symbol set ($Z = \{accept, reject\}$). This can not be simply generalized to fuzzy automata as the concepts of acceptance and rejection may imply quite different understanding in FFA_{fin}, $\text{FFA}_{no\,fin}$, and FFA with several output symbols. Nevertheless, whatever definition we use for acceptance and rejection, a level of activation (mv) will be associated with them. Therefore, to generalize the process of calculation the extent of acceptance/rejection, we introduce a new function F_3, which we call *acceptance calculation function*.

Definition 13 (*Acceptance calculation function*). In a GFA, the extent to which an input string will be accepted/rejected will be calculated by a function F_3, represented as:

$$F_3 : [0, 1]^* \to [0, 1] \tag{29}$$

Although we presented a general definition for F_3, its type and the way it operates, depends on two parameters:

- It depends on whether F_3 is being used to calculate the extent of acceptance or the extent of rejection.
- It depends on the type of the automaton, i.e. whether it is a FFA_{fin}, FFA_{nofin}, or FFA with several output symbols (FMFA or FMLFA).

Therefore, to clarify the concept of acceptance for input strings to a GFA and devise methodologies to assign mv's to them, we consider different scenarios depending on the output mapping of the GFA and whether it includes some final states or not.

Definition 14 (*Acceptance in FFA with final states*). In the FFA_{fin} $\widetilde{F} = (Q, \Sigma, \widetilde{\delta}, \widetilde{R}, Z, \omega, F_1, F_2)$ ($Q_{fin} \neq \phi$), a string of input symbols such as $x \in \Sigma^*$ is said to be acceptable by $\widetilde{F}$, if starting from an initial state, at least one of the states in the active state set of x is final. Otherwise x is said to be rejected.

$$x \text{ is accepted by } \widetilde{F} \iff \exists q_m \mid q_m \in Dom(Q_{act}(x)) \cap Q_{fin} \tag{30}$$

$$x \text{ is rejected by } \widetilde{F} \iff Dom(Q_{act}(x)) \cap Q_{fin} = \phi \tag{31}$$

It is desirable in some applications that acceptance be conditional (subject to some thresholds). In the same manner that we defined threshold derivations and active state sets, we define the *conditional acceptance/rejection* as:

Definition 15. (*Conditional acceptance/rejection*)

$$x \text{ is accepted by } \widetilde{F} \text{ subject to } \tau_1/\tau_2 \text{ threshold}$$
$$\iff \exists q_m \mid q_m \in Dom(Q_{act}(x, \tau_1/\tau_2)) \cap Q_{fin} \tag{32}$$

$$x \text{ is rejected by } \widetilde{F} \text{ subject to } \tau_1/\tau_2 \text{ threshold}$$
$$\iff Dom(Q_{act}(x, \tau_1/\tau_2)) \cap Q_{fin} = \phi \tag{33}$$

In an FMFA, however, it makes more sense to talk about the *belonging/disbelonging to an output label* rather than *acceptance/rejection*.

Definition 16 (*Belonging to an output label*). In the FMFA $\widetilde{M} = (Q, \Sigma, \widetilde{\delta}, \widetilde{R}, Z, \gamma, F_1, F_2)$, a string $x \in \Sigma^*$ is said to belong to the output label z_l, if starting from an initial state, at least one of the states in the active state set of x generates output label z_l. Otherwise x is said to disbelong to z_l.

$$x \text{ belongs to } z_l \iff \exists q_m \mid q_m \in Dom(Q_{act}(x)) \cap Q_{z_l} \tag{34}$$

$$x \text{ disbelongs to } z_l \iff Dom(Q_{act}(x)) \cap Q_{z_l} = \phi \tag{35}$$

Table 5
Acceptance/rejection for different classes of GFA

			class of GFA		
			FFA_{fin}	FMFA	FFA_{nofin}
acceptance/rejection	acceptance	name of the parameter	level of acceptance	level of belonging	degree of acceptance
		notation	$\mu_{acc}(x)$	$\mu_{z_l}(x)$	$\eta_{acc}(x)$
		working set of $F_3(S_{acc})$	$Dom(Q_{act}(x)) \cap Q_{fin}$	$Dom(Q_{act}(x)) \cap Q_{z_l}$	$Q_{act}(x)$
	rejection	name of the parameter	level of rejection	level of disbelonging	degree of rejection
		notation	$\mu_{rej}(x)$	$\mu_{\overline{z_l}}(x)$	$\eta_{rej}(x)$
		working set of $F_3(S_{rej})$	$Dom(Q_{act}(x)) - Q_{fin}$	$Dom(Q_{act}(x)) - Q_{z_l}$	not applicable
parameters	conditional acceptance	name of the parameter	conditional level of acceptance	conditional level of belonging	cond. degree of acceptance
		notation	$\mu_{acc}(x, \tau_1/\tau_2)$	$\mu_{z_l}(x, \tau_1/\tau_2)$	$\eta_{acc}(x, \tau_1/\tau_2)$
		working set of $F_3(S_{acc})$	$Dom(Q_{act}(x, \tau_1/\tau_2)) \cap Q_{fin}$	$Dom(Q_{act}(x, \tau_1/\tau_2)) \cap Q_{z_l}$	$Q_{act}(x, \tau_1/\tau_2)$
	conditional rejection	name of the parameter	conditional level of rejection	conditional level of disbelonging	cond. degree of rejection
		notation	$\mu_{rej}(x, \tau_1/\tau_2)$	$\mu_{\overline{z_l}}(x, \tau_1/\tau_2)$	$\eta_{rej}(x, \tau_1/\tau_2)$
		working set of $F_3(S_{rej})$	$Dom(Q_{act}(x, \tau_1/\tau_2)) - Q_{fin}$	$Dom(Q_{act}(x, \tau_1/\tau_2)) - Q_{z_l}$	not applicable

To save space, we have summarized the relevant parameters of acceptance and rejection for different classes of GFA in Table 5. However, the following points are noticeable:

(1) We have used different terms for the extent of acceptance/rejection in different classes of GFA, to emphasize on the diversity and different implications of the concept of acceptance in GFA.
(2) All parameters are calculated using function F_3 via Algorithm 2, where the working set of F_3 is either the set S_{acc} or S_{rej}, accordingly. S_{acc} and S_{rej} specify the sets of states, which participate in the calculation of acceptance and rejection, respectively.

Algorithm 2 (*Generic algorithm for calculating the extent of acceptance* (*rejection*) *in GFA*). In the GFA $\widetilde{F} = (Q, \Sigma, \widetilde{\delta}, R, Z, \omega, F_1, F_2)$, the extent of acceptance (rejection)

of an input string x, is the output produced by the function F_3 where its inputs are the mv's of all the states in the set S_{acc} (S_{rej}). See Table 5.

$$\chi = \overset{n}{\underset{i=1}{F_3}}(\mu(q_i)) \tag{36}$$

where:

- $q_i \in S_{acc}$ (S_{rej}),
- n is the cardinality of the set S_{acc} (S_{rej}).
- χ represents one of the parameters: level of acceptance or rejection ($\mu_{acc}(x)/\mu_{rej}(x)$), level of belonging or disbelonging ($\mu_{z_l}(x)/\mu_{\overline{z_l}}(x)$), degree of acceptance or rejection ($\eta_{rej}(x)/\eta_{rej}(x)$), or their conditionals ($\mu_{acc}(x, \tau_1/\tau_2)/\mu_{rej}(x, \tau_1/\ \tau_2)$, etc.), accordingly.

(3) Although F_3 can be used both for calculation of acceptance and rejection, we call it *acceptance calculation function* to distinguish that from F_1 (membership assignment function) and F_2 (multi-membership resolution function). Similar to F_1 and F_2, F_3 can be any reasonable function, e.g. Max, Min, Mean, etc. However, the following axioms should be satisfied by F_3:

Axiom 6. $0 \leqslant \overset{n}{\underset{i=1}{F_3}}(\mu(q_i)) \leqslant 1$.

Axiom 7. $F_3(\phi) = 0$.

Axiom 8. $\overset{n}{\underset{i=1}{F_3}}(\mu(q_i)) = a$ if $\forall i\ (\mu(q_i) = a)$.

The logic behind Axioms 7 and 8 is essentially the same as Axioms 4 and 5.

(4) FFA_{fin} have two interesting characteristics.
   (a) As can be seen from Algorithm 2, acceptance and rejection are not necessarily complementary. For example, the level of acceptance and rejection of a string may be 0.8 and 0.6, respectively. While inconsistent with a probabilistic paradigm, this is in keeping with fuzzy set theory.
   (b) The type of the function F_3, which is used to compute $\mu_{acc}(x)$ and $\mu_{rej}(x)$, can be the same or different. For example $\mu_{acc}(x)$ may be computed using Max, while $\mu_{rej}(x)$ is being computed using Min. However, for practical applications and as far as we are emphasizing on the concepts of acceptance and rejection, it is more sensible that $\mu_{acc}(x)$ and $\mu_{rej}(x)$ are computed using the same function.

(5) In the FFA_{nofin}, we can still talk about acceptance and rejection (degree of acceptance (μ_{acc}) or rejection (μ_{rej})). Calculation of $\mu_{acc}(x)$ is related to $Q_{acc}(x)$. Unlike the class of FFA_{fin}, here it is more reasonable to assume that the degree of acceptance and rejection are complementary. Therefore, once the degree of acceptance (conditional degree of acceptance) of the string x ($\eta_{acc}(x)/\eta_{acc}(x, \tau_1/\tau_2)$) is computed, its (conditional) degree of rejection can be calculated as:

$$\eta_{\text{rej}}(x) = 1 - \eta_{acc}(x) \tag{37}$$

$$\eta_{\text{rej}}(x, \tau_1/\tau_2) = 1 - \eta_{acc}(x, \tau_1/\tau_2) \tag{38}$$

(6) In all cases (FFA_{fin}, FFA_{nofin}, FMFA) for the calculation of acceptance and rejection, multi-membership resolution algorithm is applied first. In this way, all states who will contribute to the calculation of acceptance and rejection will have a single mv.
(7) It is possible to incorporate F_3 into Definition 8 (general fuzzy automaton). However, due to the diversity of the meanings and implications of acceptance and rejection in different classes of GFA, we preferred not to do that.

3.5.2. A different approach to assigning mv to a string

For some applications, it may be useful to use the following definition to assign mv to a string. It is adapted from the mv definition of strings in a Regular Fuzzy Grammar (RFG) [52,33]. This definition is sometimes called *max–min rule*:

Definition 17 (*Membership value of a string*). The mv of a string x denoted as $\mu(x)$, is the maximum membership value among all its derivations, where the mv of a derivation is the minimum transition weight encountered in that derivation.

Given that x has n derivations, and $x = a_1a_2 \ldots a_{k\ ak+1} \ldots a_m$, and that the ith derivation of x is: $der_i(x) = q_{i_0}q_{i_1} \ldots q_{i_k}q_{i_{k+1}} \ldots q_{i_m}$, where $q_{i_0} \in \widetilde{R}$, the mv of $der_i(x)$ is computed as:

$$\mu(der_i(x)) = \text{Min}\{\delta(q_{i_0}, a_1, q_{i_1}), \ldots, \delta(q_{i_k}, a_{k+1}, q_{i_{k+1}}), \ldots, \delta(q_{i_{m-1}}, a_m, q_{i_m})\}$$

And the general formula to compute the mv of x will be:

$$\mu(x) = \underset{i=1\text{ to }n}{\text{Max}}\{\mu(der_i(x))\} = \underset{i=1\text{ to }n}{\text{Max}}\{\underset{k=1\text{ to }m}{\text{Min}}\{\delta(q_{i_{k-1}}, a_k, q_{i_k})\}\} \tag{39}$$

Two points are noticeable:

(1) To find the mv of a string, we should examine all of its derivations (exhaustive search). For an ordinary size 100-state FFA with an average of five transitions from each state upon the same input symbol, a string of length 30 can have 5^{30} derivations (or even more). It takes more than 29,000 years to find the mv of such a string, assuming that each derivation can be processed in 1 ns. Therefore, finding the mv of a string using the max–min rule is in the general sense an NP-hard problem. Consequently, we have to rely on heuristic methods for this approach. We are now investigating if an efficient heuristic approach can be developed by devising appropriate choices to F_1, F_2, and possibly F_3.
(2) The max–min rule is somehow an accepted definition for the mv of a string belonging to an RFG. Based on this specific definition, for each RFG, a Moore DFA can be derived, which accepts the language of RFG (the mv of the strings are the output labels of the derived DFA) [52]. Since max–min rule is NP-hard, the algorithm developed in [52], suffers from intractability and impracticality for large FFA and long strings.

However, other options are possible for the mv of the strings of a RFG (as we discussed here). Whether DFA acceptors (Moore or Mealy) can be derived in the general sense, is not known yet and needs more investigation.

4. Experimental examples

In this section, we present some examples to show different implications of acceptance, and the capabilities achieveable by the augmented transition function $\widetilde{\delta}$ (applied through function F_1) and multi-membership resolution (applied through function F_2) and their combination.

Example 4. Consider the General Fuzzy Automaton (GFA) $\widetilde{F}$ of Fig. 5. It is defined as:

$\widetilde{F} = (Q, \Sigma, \widetilde{\delta}, \widetilde{R}, Z, \omega, F_1, F_2)$, where
$Q = \{q_0, q_1, q_2, q_3, q_4\}$: Set of states.
$\Sigma = \{a, b\} \cup \epsilon$: Set of input symbols.
$\widetilde{R} = \{(q_0, \mu^{t_0}(q_0))\} = \{(q_0, 1)\}$: Start fuzzy set.
$Z = \{accept\}$: Set of output labels. There is a single output symbol. We call it *accept*.
$\widetilde{\delta} : (Q \times [0,1]) \times \Sigma \times Q \overset{F_1(\mu,\delta)}{\rightarrow} [0,1]$: The augmented transition function.
F_1: defined in different ways as will be seen.
F_2: not applicable. There is no multi-membership.

Assume that $\widetilde{F}$ belongs to FFA_{fin} class, and all states are final ($Q_{fin} = Q$). Then, ω is defined as:

$$\omega : Q \to Z, \quad \omega(q_i) = accept \ \forall q_i \in Q.$$

Suppose the mv of the successors (next states) are computed using transition-based membership. i.e. $F_1(\mu, \delta) = \delta$. Then:

$$\mu^{t+1}(q_j) = \widetilde{\delta}((q_i, \mu^t(q_i)), a_k, q_j) = F_1(\mu^t(q_i), \delta(q_i, a_k, q_j)) = \delta(q_i, a_k, q_j)$$

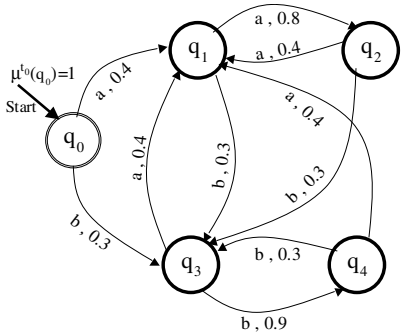

Fig. 5. The FFA of Example 4.

Table 6
The mv of different grammars in Example 4-a ($k > 0$)

Input string (x)	$Q_{act}(x)$	$\mu_{act}(x)$[a]	$\mu_{rej}(x)$[b]	$\eta_{rej}(x)$
ϵ	$\{(q_0,1.0)\}$	1.0	0.0	0.0
$(a+b)^*a^{2k-1}$	$\{(q_1,0.4)\}$	0.4	0.0	0.6
$(a+b)^*a^{2k}$	$\{(q_2,0.8)\}$	0.8	0.0	0.2
$(a+b)^*b^{2k-1}$	$\{(q_3,0.3)\}$	0.3	0.0	0.7
$(a+b)^*b^{2k}$	$\{(q_4,0.9)\}$	0.9	0.0	0.1

[a] No matter what F_3 is, $\mu_{acc}(x) = F_3(\mu(x)) = \mu(x)$ by Axiom 8, since there is only a single state in $Q_{act}(x)$.
[b] No matter what F_3 is, $\mu_{rej}(x) = F_3(\phi) = 0$ by Axiom 7, as there is no non-final states in $Q_{act}(x)$, i.e. $Q_{act}(x) - Q_{fin} = \phi$.

(a) (i) Since all states are final, all input strings are acceptable (the supported grammar is $\widetilde{G} = (a+b)^*$), but their level of acceptance may be different. However, it is trivial to verify that the mv of any input string depends on the last substring of the same symbol (i.e. a^n or b^n, $n > 0$) as shown in Table 6.

(ii) An interesting point is that, if we assume that $\widetilde{F}$ belongs to the class of FFA_{nofin}, we can still talk about acceptance and rejection of the strings (we should talk about the degree of acceptance and rejection), and the degree of acceptance ($\eta_{acc}(x)$) will be the same as level of acceptance ($\mu_{acc}(x)$), but now the degree of rejection ($\eta_{rej}(x)$) will be different from level of rejection ($\mu_{rej}(x)$), as mentioned in the fifth column of Table 6.

(iii) We see that this GFA can be interpreted as a (deterministic) Mealy machine, if we consider the transition weights as the output labels associated with the transitions. i.e. $Z = \{0.3, 0.4, 0.8, 0.9, 1.0\}$. Also, a (deterministic) Moore machine with the same number of states (five states) and with the mv's associated with the states (rather than transitions) as the output labels, can handle grammar $\widetilde{G}$.

(b) (i) Suppose that only one type of the grammars mentioned in Table 6 is acceptable. Then, we can define the corresponding state to be a final state. As a result, the GFA of Fig. 5 will now be a FFA_{fin} with one final state. For example if only the fuzzy grammar $\widetilde{G}_1 = (a+b)^*a^{2k-1}$ is acceptable, we will have:
$Z = \{accept, reject\}$ or $\{1,0\}$: Set of output labels.
$Q_{fin} = \{q_1\}$
$\omega: Q \to Z$: Output map.
$\omega(q_1) = 1$ (accept)
$\omega(q_0) = \omega(q_2) = \omega(q_3) = \omega\ (q_4) = 0$ (*reject*).
Then all acceptable strings have a level of acceptance (mv) of 0.4

$$\forall x \text{ if } q_1 \in Q_{act}(x) \Rightarrow \mu_{acc}(x) = 0.4 \tag{40}$$

While rejected strings will have different levels of rejection, as follows:

$$\begin{aligned}
&\forall x \text{ if } q_0 \in Q_{act}(x) \Rightarrow \mu_{\text{rej}}(x) = 1.0\\
&\forall x \text{ if } q_2 \in Q_{act}(x) \Rightarrow \mu_{\text{rej}}(x) = 0.8\\
&\forall x \text{ if } q_3 \in Q_{act}(x) \Rightarrow \mu_{\text{rej}}(x) = 0.3\\
&\forall x \text{ if } q_4 \in Q_{act}(x) \Rightarrow \mu_{\text{rej}}(x) = 0.9\,^{5}
\end{aligned} \tag{41}$$

(ii) In a similar manner, the acceptance of the fuzzy grammar $\widetilde{G}_1$ can be realized by a FMLFA (Fuzzy MeaLy Finite-state Automaton), with the output label mapped to all transitions leading to q_1 to be 1 (accept) and the output label mapped to all other transitions to be 0 (reject) as follows:
$Z = \{accept, reject\}$ or $\{1,0\}$: Set of output labels.
$\lambda : Q \times \Sigma \times \rightarrow Z$: Output map.
$\lambda(q_0, a, q_1) = \lambda(q_2, a, q_1) = \ldots = 1$ (*accept*).
$\lambda(q_0, b, q_3) = \lambda(q_1, a, q_2) = \lambda\ (q_2, b, q_3) = \ldots = 0$ (*reject*).
It should be noted that the level of acceptance and rejection of the strings are assigned in the same way as (b-i), i.e. Eqs. (40) and (41).

Example 5. Consider again the GFA of Fig. 5. This time, assume that it belongs to the class of FFA_{nofin}, i.e. $Q_{fin} = \phi$. Instead of transition-based membership, we use $\widetilde{\delta}$ function defined as follows:

$$\begin{aligned}
\mu^{t+1}(q_j) &= \widetilde{\delta}((q_i, \mu^t(q_i)), a_k, q_j) = F_1(\mu^t(q_i), \delta(q_i, a_k, q_j))\\
&= \begin{cases} \mu^t(q_i) + \delta(q_i, a_k, q_j) & \text{if } 0 < \mu^t(q_i) + \delta(q_i, a_k, q_j) \leqslant 1\\ \mu^t(q_i) + \delta(q_i, a_k, q_j) - 1 & \text{if } \mu^t(q_i) + \delta(q_i, a_k, q_j) > 1 \end{cases}
\end{aligned}$$

i.e. function F_1 calculates the sum of the mv of the predecessor and the weight of transition and bounds it in the interval $[0, 1]$. This bounded value is then assigned to the active state.

This mapping function makes $\widetilde{F}$ an interesting periodic fuzzy automaton. Upon periodic input strings, the active states will also be repeated with periods which are the same as the length of the strings period (we call this: *state period* (T_q)). But, the mv's associated with each state, will be different from period to period. However, the mv's will also be periodic (we call this: *mv period* (T_μ)), but their period is a multiple of the state period, i.e. $T_\mu = kT_q$, where $1 < k \leqslant 10$. Some strings together with their state periods and mv periods are illustrated in Table 7. Also, the detailed operation of the automaton upon input $(ba^3b^2a^4)^m$ $(m \geqslant 1)$ is shown in Table 8.

[5] Only when x = ϵ, $q_0 \in Q_{act}(x)$ and hence $\mu_{\text{rej}}(x) = 1.0$.

Table 7
Some strings together with their state periods (T_q) and mv periods (T_μ) for the fuzzy automaton of Example 5 ($m \geqslant 1$)

Input string	State period	T_q	T_μ
$(ab)^m$	$q_1, q_3, \ldots$	2	20
$(a^2b^2)^m$	$q_1, q_2, q_3, q_4, \ldots$	4	20
$(ba^3b^2a^4)^m$	$q_3, q_1, q_2, q_1,\ q_3, q_4, q_1, q_2, q_1, q_2, \ldots$	10	20
$(a^2b^3a^2)^m$	$q_1, q_2, q_3, q_4, q_3, q_1, q_2, \ldots$	7	70

Examples 4 and 5 illustrate GFA which have no transition overlaps, while this is a very essential property of the fuzzy automata. Next example shows the behaviour of a GFA under different combinations of F_1 and F_2.

Example 6. Consider the FFA in Fig. 6 with several transition overlaps. It is specified as:

$\widetilde{F} = (Q, \Sigma, \widetilde{\delta}, \widetilde{R}, Z, \omega, F_1, F_2)$ where
$Q = \{q_0, q_1, q_2, q_3, q_4\}$: Set of states.
$\Sigma = \{a, b\}$: Set of input symbols.
$\widetilde{R} = \{(q_0, \mu^{t_0}(q_0))\} = \{(q_0, 1)\}$
$Z = \phi$.
ω: not applicable.
$\widetilde{\delta} : (Q \times [0,1]) \times \Sigma \times Q \overset{F_1(\mu,\delta)}{\rightarrow} [0,1]$: The augmented transition function.
F_1,F_2: varying as shown in Table 9.

Note that in this example, $Q_{fin} = \phi$. The operation of this fuzzy automaton upon input string "a^2b^3a" is shown in Table 9 for different membership assignment functions and multi-membership resolution strategies. In this table, we have considered different cases for combining functions F_1 and F_2. Of course not all these combinations are reasonable for this specific example (e.g. Cases 4 and 5 make all mv's 1.0). This is happening due to the small size of this fuzzy automaton which is not a practical FFA. The goal of this example however, is to show the capabilities achievable by combining membership assignment function (F_1) and multi-membership resolution function (F_2).

5. Conclusion, discussion, and future work

In our efforts to use fuzzy automata as a modeling tool for some applications, we discovered that the theoretical background of fuzzy automata is not established well enough to define operational characteristics. In particular, the following issues seemed to be of high priority to be elaborated for a better established background:

Table 8
Operation of the automaton of Example 5 upon input $(ba^3b^2a^4)^m$, where $m \geqslant 1$. For this string: $T_q = 10$ and $T_\mu = 20$

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	...
Input	b	a	a	a	b	b	a	a	a	a	b	a	a	a	b	b	a	a	a	a	...	...
State	q_3	q_1	q_2	q_1	q_3	q_4	q_1	q_2	q_1	q_2	q_3	q_1	q_2	q_1	q_3	q_4	q_1	q_2	q_1	q_2	...	...
mv	0.3	0.7	0.5	0.9	0.2	0.1	0.5	0.3	0.7	0.5	0.8	0.2	1.0	0.4	0.7	0.6	1.0	0.8	0.2	1.0	...	...

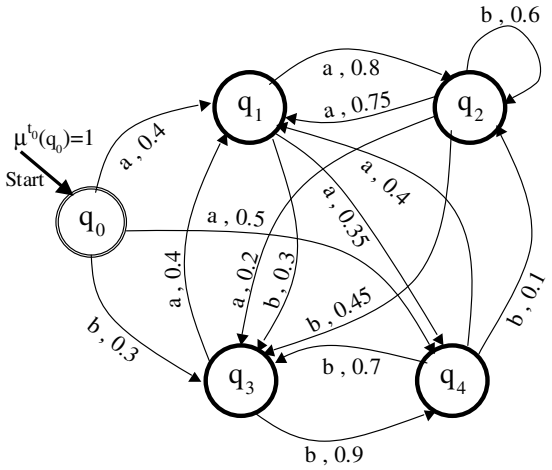

Fig. 6. The GFA of Example 6.

(1) Transition-based membership which was previously more or less the common method for assigning mv's to active states of a fuzzy automaton, has a serious drawback. The mv's of the predecessors will not be considered in assigning the activation level (mv) to successors. Specifically, there is no way to incorporate the mv of the initial fuzzy states into their successors which makes the fuzziness or nonfuzziness of the start states a moot point to discuss.
(2) Multi-membership is an essential and natural characteristic of fuzzy automata. However, for operational purposes, it is desirable to resolve the multi-membership, such that a single value can be attributed to each active state. A very important point is that the resolution strategy is preferred not to modify the structure of the FFA under consideration, i.e. no extra state and/or transition should be created.
(3) The significance of the output mapping and presence or absence of final states was not clearly defined in the literature.
(4) Another important insufficiency of the current literature is the lack of methodologies which enable us to define and analyze the continuous operation of fuzzy automata. There are many questions with no specific and well-developed answers. Questions such as: how do we refer to the set of states which are active at any time? In case we need a crisp decision as the final output of an FFA, how do the active states contribute to this decision? What do concepts of acceptance and rejection mean in a fuzzy automaton? How can we relate them to the final (and possibly crisp) output of an FFA? What is the significance of final states in an FFA?

5.1. Contributions

We devised a new methodology called *augmented transition function* ($\widetilde{\delta}$), which incorporates both the weight of the transition and the mv of a predecessor to calcu-

Table 9
Active states and their mv's at different times in Example 6

time	0	1		2			3	
input	ϵ	a		a			b	
$Q_{act}(t)$	q_0	q_1	q_4	q_1	q_2	q_4	q_2	q_3
mv^1	1.0	0.4	0.5	0.4	0.8	0.35	0.1	0.3
mv^2	1.0	0.4	0.5	0.4	0.4	0.35	0.1	0.3
mv^3	1.0	0.4	0.5	0.4	0.4	0.35	0.4	0.4
mv^4	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
mv^5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
mv^6	1.0	0.4	0.5	0.4	0.4	0.35	0.25	0.35
mv^7	1.0	0.7	0.75	0.575	0.75	0.525	0.763	0.613

4			5			6	
b			b			a	
q_2	q_3	q_4	q_2	q_3	q_4	q_1	q_3
0.6	0.45	0.9	0.1	0.45	0.9	0.4	0.2
0.1	0.1	0.3	0.1	0.1	0.1	0.1	0.1
0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.2
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
0.25	0.25	0.35	0.175	0.3	0.25	0.242	0.175
0.682	0.607	0.757	0.641	0.729	0.754	0.696	0.421

$$^1F_1(\mu,\delta) = \delta\ , \qquad F_2() = \mu^{t+1}(q_m) = \underset{i=1\ to\ n}{Min}[F_1(\mu^t(q_i), \delta(q_i, a_k, q_m))]$$

$$^2F_1(\mu,\delta) = Min(\mu,\delta)\ , \qquad F_2() = \mu^{t+1}(q_m) = \underset{i=1\ to\ n}{Min}[F_1(\mu^t(q_i), \delta(q_i, a_k, q_m))]$$

$$^3F_1(\mu,\delta) = Min(\mu,\delta)\ , \qquad F_2() = \mu^{t+1}(q_m) = \underset{i=1\ to\ n}{Max}[F_1(\mu^t(q_i), \delta(q_i, a_k, q_m))]$$

$$^4F_1(\mu,\delta) = Max(\mu,\delta)\ , \qquad F_2() = \mu^{t+1}(q_m) = \underset{i=1\ to\ n}{Min}[F_1(\mu^t(q_i), \delta(q_i, a_k, q_m))]$$

$$^5F_1(\mu,\delta) = Max(\mu,\delta)\ , \qquad F_2() = \mu^{t+1}(q_m) = \underset{i=1\ to\ n}{Max}[F_1(\mu^t(q_i), \delta(q_i, a_k, q_m))]$$

$$^6F_1(\mu,\delta) = Min(\mu,\delta)\ , \qquad F_2() = \mu^{t+1}(q_m) = [\sum_{i=1}^{n} F_1(\mu^t(q_i), \delta(q_i, a_k, q_m))]\ /\ n$$

$$^7F_1(\mu,\delta) = \tfrac{\mu+\delta}{2}\ , \qquad F_2() = \mu^{t+1}(q_m) = \underset{i=1\ to\ n}{Max}[F_1(\mu^t(q_i), \delta(q_i, a_k, q_m))]$$

n : The number of simultaneous (overlapping) transitions to state q_m.

late the mv of a successor. The membership assignment function $F_1(\mu,\delta)$ which is applied in association with $\widetilde{\delta}$, can vary from application to application. However, the

transition-based membership can be considered as a special case of the augmented transition function.

Multi-membership may happen both for final states of a FFA (if applicable) or non-final states. For operational reasons, we assume that every multi-membership state be resolved and a unified mv be attributed to that. We introduced a new function F_2 called *multi-membership resolution function*, which specifies the strategy which should be applied to multi-membership states. An interesting point about the process of multi-membership resolution as described here, is that, it can be considered as a primary step for defuzzification in FFA. Defuzzification as we know, is the process often done as the last step in fuzzy systems to produce a final crisp value. But, in multi-membership resolution, the final result which is a single mv, is still fuzzy. In other words, multi-membership resolution is in fact membership unification. However, as most applications need to produce crisp results as the output, we have to defuzzify the results produced by a FFA. No matter if the FFA has output mapping or not, the results are usually embedded in the mv's of the active states (final or non-final), all or some of which may be multi-membership. Since the defuzzification should be done on these active states, a single mv has to be associated with each of them, which is done by multi-membership resolution. In fact, multi-membership resolution can be considered as a primary process for defuzzification. That is why we suggest multi-membership resolution process to be called *predefuzzification* in the realm of fuzzy automata.

A new general definition for a fuzzy automaton was presented (Definition8—GFA). We incorporated the membership assignment function (F_1) and multi-membership resolution function (F_2) as two additional parameters into this new definition. These two parameters should be specified by the user together with other required parameters of a fuzzy automaton such as states, input symbols, transition weights, etc. F_1 and F_2 can be the same or different depending on the application.

Although, the generality of fuzzy automata has been the subject of several research efforts [38,37,40,41,15], to the best of our knowledge, GFA as introduced here, is the most general formulation of fuzzy automata presented so far, at least in the realm of discrete spaces. Moreover, we do believe that Definition 8 can be used as the definition of a *general automaton*, which degenerates to other types of automata under various restrictions. In fact GFA encompasses not only other types of automata, but also several other computational paradigms [39]. Even so, the generality of GFA is so motivating and challenging that deserves lots of future research.

Regarding the significance of output mapping, we introduced the Fuzzy Moore Finite-state Automata (FMFA) and the Fuzzy MeaLy Finite-state Automata (FMLFA) for the applications where there are several output labels. While in FMFA outputs are associated with the states, in a FMLFA they are associated with the transitions. FMFA seem to be simpler and closer to justification and for the class of our applications we will rely on them whenever we deal with multiple outputs. Nevertheless, FMLFA are a challenging and open issue and seem to have many interesting characteristics. For example, since in FMFA, the output is associated with the state, it is reasonable that an output label inherits everything from the corresponding state,

e.g. the mv, the multi-membership resolution strategy, etc. But, generally in FMLFA, we can have more options. For example, to assign a mv to an output label, any of the following parameters or a combination of them can be used:

- The weight of the corresponding transition.
- The (resolved) mv of the predecessor (source state of the transition).
- The (resolved) mv of the successor (destination state of the transition).

We also believe that some of the concepts introduced here (active state set, derivation, acceptance, etc.) should be revised to become applicable to a general FMLFA. It is likely that we will see more research on FMLFA whose capabilities are yet to be discovered.

The FFA with final states are of great importance in practice, as in most applications we have to rely on some final states for final decision making. The following issues should be mentioned in this regard:

(1) The FFA with final states are specific cases of FMFA where we have a binary set of output labels, i.e. $Z = \{accept, reject\}$.
(2) It is understood that the concepts of acceptance and rejection should not be necessarily complemenetary in FFA. This is particularly the case for the class of FFA with final states (FFA_{fin}). We assigned the terms *level of acceptance* ($\mu_{acc}(x)$) and *level of rejection* ($\mu_{rej}(x)$) to refer to the acceptance and rejection of an input string x, respectively. The generic Algorithm 2 which is used to compute the extent of acceptance/rejection in different classes of GFA, calculates $\mu_{acc}(x)$ and $\mu_{rej}(x)$ for the input strings in the class of FFA_{fin}. The calculation of $\mu_{acc}(x)$ is related to the active states which are final, while the calculation of $\mu_{rej}(x)$ is related to the active states which are non-final. This method of calculating $\mu_{acc}(x)$ and $\mu_{rej}(x)$ is in fact an emphasis on the interesting and challenging issue that acceptance and rejection are not necessarily complementary in FFA_{fin}.
(3) For applicational reasons, we preferred that in the class of FFA without final states (FFA_{nofin}), acceptance and rejection be complementary. To distinguish the concept of acceptance and rejection in this class from those in the class of FFA_{fin}, we used the terms *degree of acceptance* ($\eta_{acc}(x)$) and *degree of rejection* ($\eta_{rej}(x)$) to refer to the acceptance and rejection of a string x, respectively. The calculation of $\eta_{acc}(x)$ is related to the active state set.
However, the class of FFA_{nofin} can still be considered as another specific case of FMFA with a single output label. It can be called anything, although *accept* is more consistent with the conventional spectrum of automata.
(4) Calculation of the extent of acceptance/rejection is done by a function F_3, which we called *acceptance calculation function*. F_3 should also be specified based on the application and the class of FFA, i.e. whether it is FFA_{fin}, FFA_{nofin}, FMFA, or FMLFA. For the class of FFA_{fin}, where acceptance and rejection are not necessarily complementary (and more generally for FMFA and FMLFA), F_3 can be the same or different for the calculation of acceptance and rejection depending on the requirements of the application.

A very interesting and challenging implication of our approach is that a zero-weight transition is possible and is different from no transition. A zero-weight transition may give rise to the activation of a successor due to the activation of its predecessor. This challenging issue is now under more development and we will have more to say about its usefulness in future.

As mentioned previously, the key motivation of this work was the insufficiency of the current literature to handle the applications which rely on fuzzy automata as a modeling tool. It will be interesting to see how the developed concepts and algorithms can be used in practice. Currently, we are also working on some applications which can be modelled by FFA. The achieved results and how these models can be learned by neural networks, will be reported in forth-coming papers.

To say the final word, we do believe that this is just a starting work to enrich the ground of fuzzy automata and make this appealing tool more applicational and useful.

Acknowledgments

The authors would like to thank anonymous reviewers whose highly valuable and constructive comments improved the manuscript considerably. Professor Kremer is funded by grants from NSERC, CFI, ORDCF, and OIT, to whom we would like to present our greetings.

References

[1] M.A. Arbib, From automata theory to brain theory, Int. J. Man–Machine Stud. 7 (3) (1975) 279–295.
[2] W.R. Ashby, Design for a brain, Chapman and Hall, London, 1954.
[3] A.W. Burks, Logic, biology and automata—some historical reflections, Int. J. Man–Machine Stud. 7 (3) (1975) 297–312.
[4] W.S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, Bull. Math. Biophys. 5 (1943) 115–133.
[5] M.L. Minsky, Computation: Finite and Infinite Machines, Prentice-Hall, Englewood Cliffs, NJ, 1967, Chapter 3, pp. 32–66.
[6] A. Turing, On computable numbers, with an application to the entscheidungs problem, Proc. London Math. Soc. 42 (1936–37) 220–265.
[7] J. von Neumann, Theory of Self-Reproducing Automata, University of Illinois Press, Urbana, 1966.
[8] B.R. Gaines, L.J. Kohout, The logic of automata, Int. J. Gen. Syst. 2 (1976) 191–208.
[9] D. Ashlock, A. Wittrock, T. Wen, Training finite state machines to improve PCR primer design, in: Proceedings of the 2002 Congress on Evolutionary Computation (CEC'02), 2002.
[10] R. Maclin, J. Shavlik, Refining domain theories expressed as finite-state automata, in: L.B.G. Collins (Ed.), Proceedings of the 8th International Workshop on Machine Learning (ML'91), Morgan Kaufmann, San Mateo CA, 1991.
[11] R. Maclin, J. Shavlik, Refining algorithm with knowledge-based neural networks: improving the chou-fasma algorithm for protein folding, in: S. Hanson, G. Drastal, R. Rivest (Eds.), Computational Learning Theory and Natural Learning Systems, MIT Press, Cambridge, MA, 1992.
[12] B. Tucker (Ed.), The Computer Science and Engineering Handbook, CRC Press, Boca Raton, FL, 1997.

[13] L.A. Zadeh, The concept of state in system theory, in: M. Mesarovic (Ed.), Views on General System Theory, John Wiley, New York, 1964, pp. 39–50.
[14] L.A. Zadeh, The concept of system, aggregate and state in system theory, in: L.A. Zadeh, E. Polak (Eds.), System Theory, McGraw Hill, New York, 1969, pp. 3–42.
[15] J. Virant, N. Zimic, Fuzzy automata with fuzzy relief, IEEE Trans. Fuzzy Syst. 3 (1) (1995) 69–74.
[16] L.A. Zadeh, Fuzzy sets, Inform. Control 8 (1965) 338–353.
[17] G.J. Klir, B. Yuan, Fuzzy Sets and Fuzzy Logic, Theory and Applications, Prentice Hall, Englewood Cliffs, NJ, 1995.
[18] T.L. Hardy, Multi-objective decision-making under uncertainty fuzzy logic methods, Tech. Rep. TM 106796, NASA, Washington D.C., 1992.
[19] L.A. Zadeh, Fuzzy language and their relation to human and machine intelligence, Technical Report ERL-M302, Electrical Research Laboratory, University of California, Berkeley, 1971.
[20] P. Bonissone, V. Badami, K. Chiang, P. Khedkar, K. Marcelle, M. Schutten, Industrial applications of fuzzy logic at general electric, Proc. IEEE 83 (3) (1995) 450–465.
[21] S. Chiu, S. Chand, D. Moore, A. Chaudhary, Fuzzy logic for control of roll and moment for a flexible wing aircraft, Control Syst. Mag. 11 (4) (1991) 42–48.
[22] J. Corbin, A fuzzy logic based financial transaction system, Embedded Syst. Program. 7 (12) (1994) 24.
[23] D. Dubois, H. Prade, Fuzzy Sets and Fuzzy Systems, Theory and Applications, Academic Press, New York, 1980.
[24] M. Jamshidi, N. Vadiee, T.J. Ross (Eds.), Fuzzy Logic and Control, Software and Hardware Applications, Prentice-Hall, Englewood Cliffs, NJ, 1993.
[25] M. Jamshidi (Ed.), Large Scale Systems: Modelling, Control, and Fuzzy Logic, Prentice-Hall, Englewood Cliffs, NJ, 1995.
[26] M. Jamshidi, A. Tilti, L.A. Zadeh, S. Boverie (Eds.), Applications of Fuzzy Logic, Toward High Machine Intelligence Quotient Systems, Prentice-Hall, Englewood Cliffs, NJ, 1997.
[27] W.J.M. Kickert, H. van Nauta Lemke, Application of a fuzzy controller in a warm water plant, Automatica 12 (4) (1976) 301–308.
[28] C. Lee, Fuzzy logic in control systems: Fuzzy logic controllers, IEEE Trans. Syst., Man, Cybernet. SMC-20 (2) (1990) 404–435.
[29] C. Papis, E. Mamdani, A fuzzy logic controller for a traffic junction, IEEE Trans. Syst., Man, Cybernet. SMC-7 (10) (1977) 707–717.
[30] A. Torralba, J. Chavez, L. Franquelo, Fasy: a fuzzy logic based tool for analog synthesis, IEEE Trans. Comput.-Aided Design Integr. Circuits 15 (7) (1996) 705.
[31] X. Yang, G. Kalambur, Design for machining using expert system and fuzzy logic approach, J. Mater. Eng. Perform. 4 (5) (1995) 599.
[32] S. Mensch, H. Lipp, Fuzzy specification of finite-state machines, in: L. Collins (Ed.), Proceedings of the 8th International Workshop on Machine Learning (ML'91), Morgan Kaufmann, San Mateo, 1991, p. 66.
[33] J.N. Mordeson, D.S. Malik, Fuzzy Automata and Languages, Theory and Applications, Chapman and Hall/CRC, London/Boca Raton, FL, 2002.
[34] A. Pathak, S. Pal, Fuzzy grammars in syntactic recognition of skeletal maturity from X-rays, IEEE Trans. Syst. Man Cybernet. 16 (5) (1986) 657–667.
[35] H. Senay, Fuzzy command grammars for intelligent interface design, IEEE Trans. Syst. Man Cybernet. 22 (5) (1992) 1124–1131.
[36] J.R. Garitagoitia, J.R.G. de Mendivil, J. Echanobe, J.J. Astrain, F. Farina, Deformed fuzzy automata for correcting imperfect strings of fuzzy symbols, IEEE Trans. Fuzzy Syst. 11 (3) (2003) 299–310.
[37] W. Pedrycz, A. Gacek, Learning of fuzzy automata, Int. J. Computat. Intell. Appl. 1 (1) (2001) 19–33.
[38] C. Cattaneo, P. Flocchini, G. Mauri, C.Q. Vogliotti, N. Santoro, Cellular automata in fuzzy backgrounds, Physica D 105 (1997) 105–120.
[39] M. Doostfatemeh, S.C. Kremer, A fuzzy finite-state automaton that unifies a number of other popular computational paradigms, in: Proceedings of the ANNIE 2003 Conference (ANNIE 03'), ASME Press, New York, 2003.

[40] C.A. Reiter, Fuzzy automata and life, Complexity 7 (3) (2002) 19–29.
[41] A.K. Srivastava, S.P. Tiwari, A topology for fuzzy automata, in: N.R. Pal, M. Sugeno (Eds.), 2002 AFSS International Conference on Fuzzy Systems, Proceedings, Lecture Notes in Artificial Intellegence, Springer-Verlag, Berlin, 2002, pp. 485–491.
[42] M. Ying, A formal model of computing with words, IEEE Trans. Fuzzy Syst. 10 (5) (2002) 640–652.
[43] A. Blanco, M. Delgado, M.C. Pegalajar, Fuzzy automata induction using neural network, Int. J. Approx. Reason. 27 (2001) 1–26.
[44] A. Blanco, M. Delgado, M. Pegalajar, Fuzzy grammar inference using neural networks, Tech. rep., Department of Computer Science and Artificial Intelligence, University of Granada, Spain, 1995.
[45] E. Kosmatopoulos, M. Christodoulou, Neural networks for identification of fuzzy dynamical systems: an application to identification of vehicle highway systems, Technical Report, Department of Electrical and Computer Engineering, Technical University of Crete, Greece, 1995.
[46] W. Omlin, C.L. Giles, K.K. Thornber, Equivalence in knowledge representation: automata, rnns, and dynamical fuzzy systems, Proc. IEEE 87 (9) (1999) 1623–1640.
[47] W. Omlin, K.K. Thornber, C.L. Giles, Fuzzy finite-state automata can be deterministically encoded into recurrent neural networks, IEEE Trans. Fuzzy Syst. 5 (1) (1998) 76–89.
[48] G.D. Bruce, K.S. Fu, A model for finite-state probabilistic systems, in: Proceedings of the of 1st Annual Allerton Conference on Circuit and Systems Theory, 1963.
[49] J. Hopcroft, J. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, MA, 1979.
[50] Z. Kohavi, Switching and Finite Automat Theory, McGraw-Hill, New York, 1978.
[51] E. Santos, Maximin automata, Inform. Control 13 (1968) 363–377.
[52] M.G. Thomason, P.N. Marines, Deterministic acceptors of fuzzy languages, IEEE Trans. Systems, Man, Cybernet. 4 (3) (1974) 228–230.
[53] R.R. Yager, On a general class of fuzzy connectives, Fuzzy Sets Syst. 4 (3) (1980) 235–242.
[54] M. Doostfatemeh, S.C. Kremer, Representing generalized fuzzy automata in recurrent neural networks, in: Proceedings of the 17th IEEE Canada Conference (CCECE 2004, Niagara Falls), IEEE Press, New York, 2004.
[55] R. Belohlavek, Determinism and fuzzy automata, Inf. Sci. 143 (2002) 205–209.
[56] J. Mockor, Fuzzy and non-deterministic automata, Soft Comput. 3 (1999) 221–226.
[57] J. Mockor, Semigroup homomorphisms and fuzzy automata, Soft Comput. 6 (2002) 422–427.
[58] M. Doostfatemeh, S.C. Kremer, The significance of output mapping in fuzzy automata, in: Proceedings of the 12th Iranian Conference on Elecectrical Engineering (ICEE 2004), Ferdowsi University, Iran, 2004.
[59] G.H. Mealy, A method for synthesizing sequential circuits, Bell Sys. Tech. J. 34 (1955).
[60] E.F. Moore, Gedanken experiments on sequential machines, in: Automata Studies, Princeton University Press, 1956, pp. 129–153.