The Intelligence Crafter: a Fuzzy State Machine Builder Program

by F. Martin McNeill

Fuzzy Systems Engineering

619-748-7384

Fuzzy State Machines (FSM) are fuzzy generalizations of crisp state machines (CSM). The programs that build source code implementations of crisp state table representations are commercially available. A fuzzy state table representation with the associated fuzzy source code build is the purpose of this presentation. The discussion presents a working program that allows fuzzy states and fuzzy events in the table and C function build. The discussion will invite suggestions as to the general use of the FSM and possible methods of representation and encoding.

- Fuzzy Automata
- Dynamic System
- Input Events
- Output Actions
- Present States
- Next States

From Klir and Yuan, 1995: "A finite automaton (also called a finite-state machine or sequential machine) is a dynamic system operating in discrete time that transforms sequences of input states (stimuli) received at the input of the system to sequences of output states (responses) produced at the output of the system. The sequences may be finite or countably infinite. The transformation is accomplished by the concept of a dynamically changing internal state. At each discrete time, the response of the system is determined on the basis of the received stimulus and the internal state of the system. At the same time, a new internal state is determined, which replaces its predecessor. The new internal state is stored in the system to be used the next time. An automaton is called a fuzzy automaton when its states are characterized by fuzzy sets, and the production of responses and next states is facilitated by appropriate fuzzy relations."

The method discussed in Klir and Yuan has the Events(stimuli), States(internal states) and Actions(responses) distributed over all of each at each discrete time. This may be called the distributed method. The Klir and Yuan also calculates the new State truth values and calculates the Action truth values. I call this the <u>distributed</u> method.

Another method has singleton values on the Events, States and Actions. The singleton usage here is different than in fuzzy estimation transform construction. Here it means only one of each Event, State and Action has a nonzero truth value. It might be better to call this the <u>lumped</u> method. With the lumped method the truths are assigned or locally calculated(Brubaker,1991).

Here are the Klir and Yuan calculation techniques on the distributed method:.

Definitions

(Derived from Klir and Yuan, 1995) A *finite fuzzy automaton*, F_A, is a fuzzy relational system defined by the quintuple

$$F_A = (E, A, S, R, T),$$

where

E is a nonempty finite set of input Events ,
A is a nonempty finite set of output Actions,
S_P is a nonempty finite set of present States,
S_N is a nonempty finite set of next States,
R (Response relation) is a fuzzy relation on $[S_P][A]$, and
T (state-Transition relation) is a fuzzy relation on $[E][S_P][S_N]$.

Assume that $E = \{e_1, e_2, \ldots e_n\}, A = \{a_1, a_2, \ldots a_m\}, S_P = \{s_{P1}, s_{P2}, \ldots s_{Pq}\}$, and $S_N = \{s_{N1}, s_{N2}, \ldots s_{Nq}\}$, and let E^t, A^t, S_P^t, S_N^t denote the fuzzy sets that characterize, respectively, the input Events, output Actions, present State, and next State of the automaton at time t..

The idea of a fuzzy automaton is depicted in Fig. 1. Given E^t and S_P^t at some time t, fuzzy relations R and P allow us to determine A^t and S_N^t. A fuzzy set S_P^t, which characterizes the initial internal state, must be given to make the fuzzy automaton operate. Then, $S_P^t = S_N^{t-1}$ for each time $t \in N - (1)$. The equation $S_P^t = S_N^{t-1}$ is assumed to be implemented by the block called storage in Fig. 1. Its role is to store the produced fuzzy set S_N^t at each time t and release it the next time t + 1 under the label S_P^{t+1}.

Given a sequence $E^1, E^2, \ldots$, and an initial characterization S_P^1 of the internal state, fuzzy relations R and T allow us to generate the corresponding sequences $A^1, A^2, \ldots$ and $S_P^2 = S_N^1, S_P^3 = S_N^2, \ldots,$. Due to the roles of relations R and T, it is reasonable to call R a Response relation and T a state-Transition relation.

Assuming the standard fuzzy set operations, the a fuzzy automaton may operate as follows.

STATE MACHINE

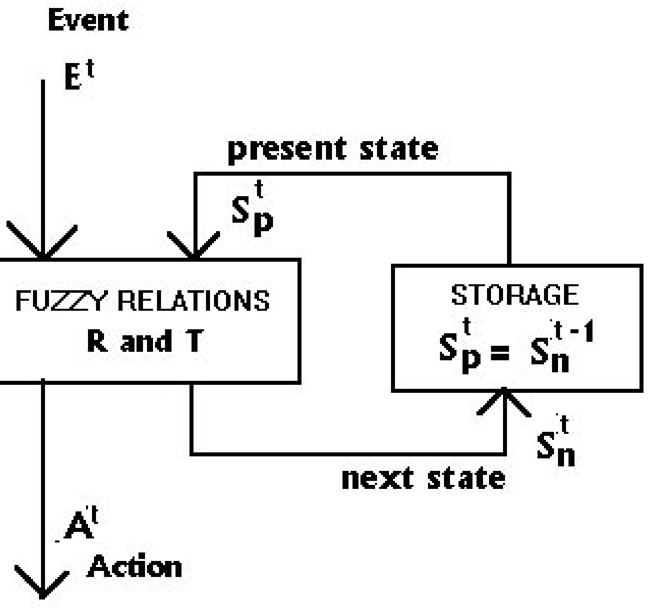


Fig. 1

For any given fuzzy input event E^t, the ternary state-Transition relation T is converted into a binary relation, T_{E^t}, on $[S_P][S_N]$ by the formula

$$T_{E^t}(s_{Pi},\ s_{Nj}) \ = \ \max_{k \in N_n}(\min[E^t(e_k),\ T(e_k,\ s_{Pi},\ s_{Nj})]). \qquad (1)$$

Then, assuming the present fuzzy state S_P^t is given, the fuzzy next state S_N^t and fuzzy output action A^t are determined by the max-min compositions

$$S_N^t \ = \ S_P^t \circ T_{E^t}, \qquad (2)$$
$$A^t \ = \ S_P^t \circ R. \qquad (3)$$

Equations (1-3) are sufficient for handling sequences of fuzzy states. Consider, for example, a sequence $E^1,\ E^2,\ldots,E^r$ of r fuzzy input events applied to a given initial fuzzy state S_P^1. Then, the fuzzy automaton produces the sequence of fuzzy internal states and the corresponding sequence of fuzzy output actions

$$\begin{array}{ll} S_N^1 \ = \ S_P^1 \circ T_{E^1}, & A^1 \ = \ S_P^1 \circ R, \\ S_N^2 \ = \ S_P^2 \circ T_{E^2}, , & A^2 \ = \ S_N^1 \circ R, \\ \ldots\ldots\ldots\ldots & \ldots\ldots\ldots\ldots \\ S_N^r \ = \ S_N^{r-1} \circ T_{E^r} & A^r \ = \ S_N^{r-1} \circ R. \end{array}$$

•Distributed Fuzzy Automata
•Meaningful Fuzzy Automata
•Lumped Automata
•Classical Crisp Automata

To define meaningful fuzzy automaton, relations R and T cannot be arbitrary. For example, some next internal state and some output action must be assured for any given input event and present internal state. When these requirements are satisfied, we call the relations deterministic. The intelligence to be crafted into the automata is in the defining of these relations R and T. With the distributed method defining the relationships is difficult. Any tool in support of this crafting would be of value. The work by Endad Khan is toward this purpose (Khan and Unal, 1994)..

Defining the relations for a lumped automata is more intuitively possible. At each Action the next State and its truth is defined from authority, ready for use when the next Event comes along. The next Event truths may also be defined at this time. The defined values may be by assignment or by local calculations. The situation at each lumped discrete time is relatively simplistic so hopefully intuitively manageable.

Here is an interesting nomenclature aside: If the count of the Events is 1 the machine is a fuzzy sequencer. If the count of the States is 1 the machine is a fuzzy decision table. With Events and States greater than one the machine is a fuzzy finite state machine. These machines in program form become fuzzy finite state programs.

•Singleton and Crisp
•Singleton and Fuzzy
•Mealy or Moore
•Program Language

When all events, states and actions of a fuzzy automaton are defined as crisp sets and R, T are crisp relations, we obtain a crisp automation, which, in general, is nonderterministic. When, in addition, all states are singletons taken from sets E, A, S (singleton means all but one are functionally zero) and relations R, T are deterministic, we obtain the classical deterministic automaton of the Mealy or Moore type. Mealy if the actions are defined as part of the state-Transition relation. Moore if the actions are defined as part of the present State.

The finite Mealy or Moore state program is an extension of automata wherein we are interested in the sequential execution of programmatic actions. The events and states are crisp and singletons with a program action at each [event][state] intersection. In the fuzzy finite state program the events and states may be fuzzy though still singletons. The machine is implemented in desired program languages.

The concept of a fuzzy automaton is a broad one, which subsumes classical crisp automata.

The Intelligence Crafter

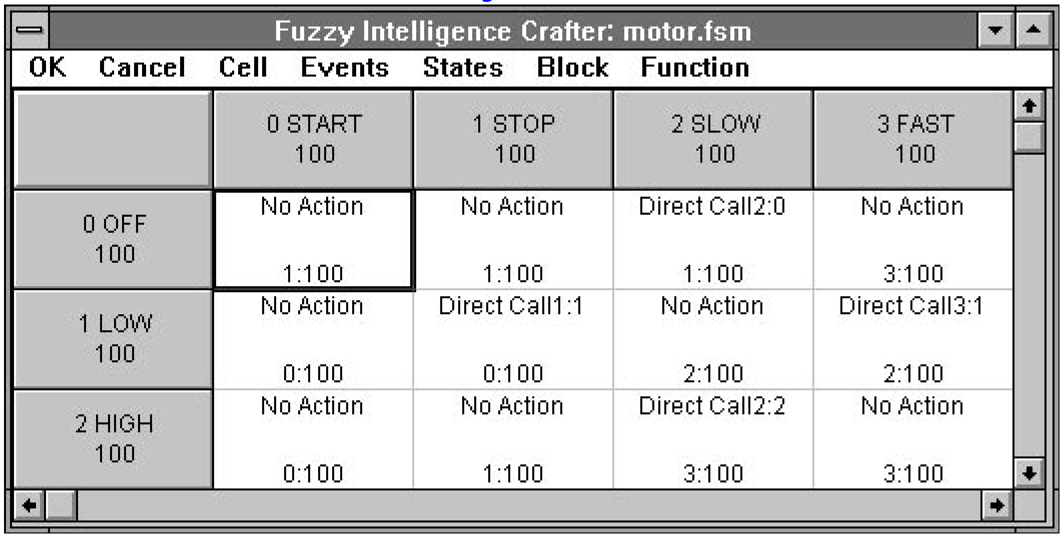
- Fuzzy State Table Editor
- C Language Build
- Assembly Language Build

The Intelligence Crafter is a Windows based editable fuzzy state table and source language compiler. The approach is similar to the fully crisp *Compeditor Finite State Compiler* by AYECO Inc., Orlando,FL. The Intelligence Crafter is a Windows program where the Compeditor is console interface.

After editing, the Intelligence Crafter will build a fuzzy finite state program that will run the machine in your program environment. Any program language is possible. C is the first supported language. Microprocessor assembly language is the second target language.

The Intelligence Crafter is a 32 bit program. Thus it requires WinNT or Windows 95 as an operating system. Win32s is not sufficient.

• Fuzzy State Table


Fuzzy Intelligence Crafter: motor.fsm
OK Cancel Cell Events States Block Function

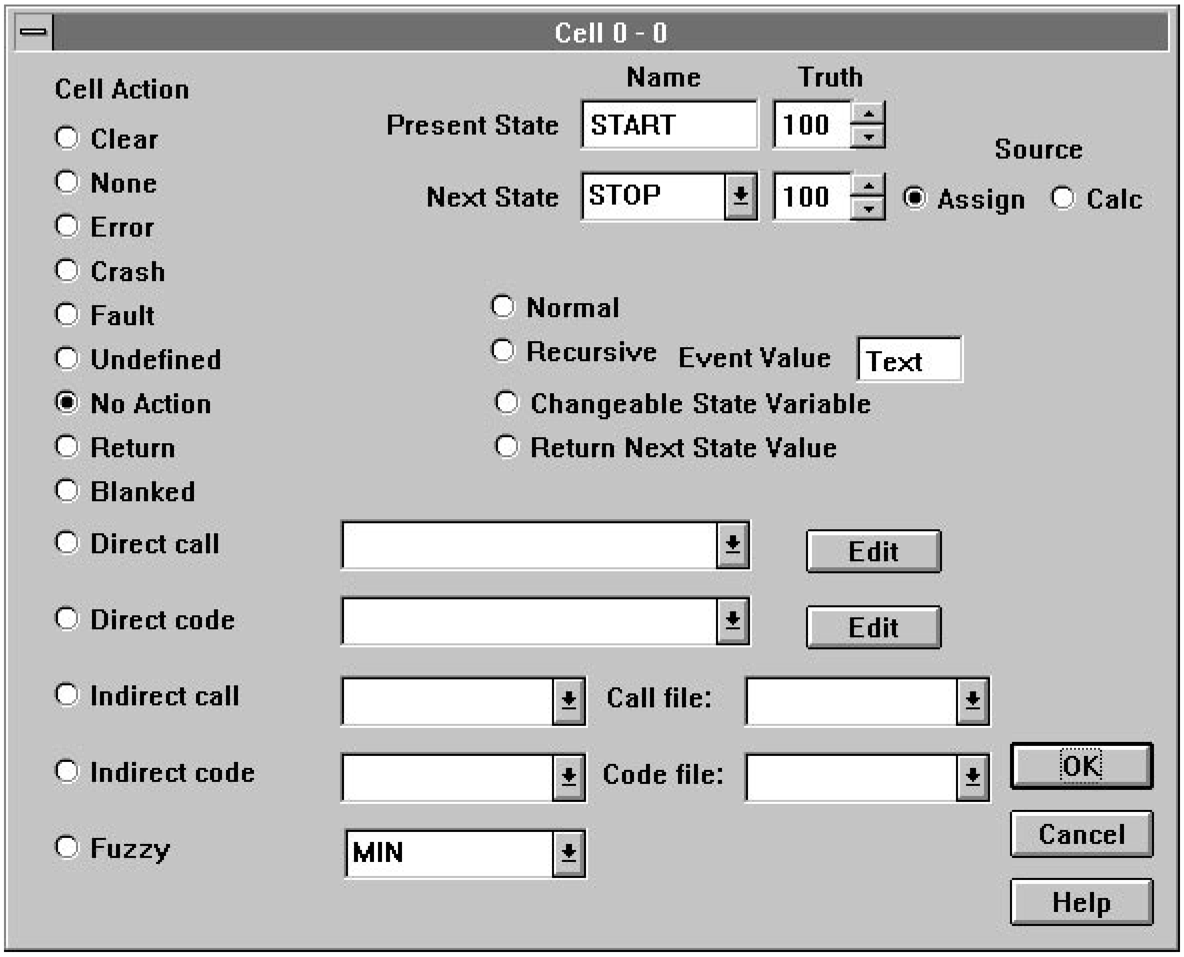
	0 START 100	1 STOP 100	2 SLOW 100	3 FAST 100
0 OFF 100	No Action 1:100	No Action 1:100	Direct Call2:0 1:100	No Action 3:100
1 LOW 100	No Action 0:100	Direct Call1:1 0:100	No Action 2:100	Direct Call3:1 2:100
2 HIGH 100	No Action 0:100	No Action 1:100	Direct Call2:2 3:100	No Action 3:100

A matrix table view of the needed states, events and actions is provided. States in a row across the top, events in a column on the left and transition relations in the intersecting matrix cells. Action outputs may occur at states or transitions.

The Intelligence Crafter fuzzy state table editor supports from 1 to 256 States and 1 to 256 Events. The maximum Action cell is thus 256 X 256 = 65536. The view is of a scrollable spreadsheet. A complete set of spreadsheet matrix editing tools is provided. The tool will evolve as wants and desires are received from users. The goal is to provide the several lumped state machine architectures provided by the Compeditor with the addition of fuzziness. If it can be seen how to meaningfully define a distributed fuzzy state machine this architecture will be supported also.

A state table debugger allows the designer to exercise and simulate the fuzzy finite state program's operation.

. Matrix Cell Editor
. Event and State Editor



Cell 0 - 0

Cell Action: Clear [✓ No Action] None Error Crash Fault Undefined Return Blanked Direct call Direct code Indirect call Indirect code Fuzzy

	Name	Truth
Present State	START	100
Next State	STOP	100

Source: [✓ Assign] Calc

Normal
Recursive Event Value Text
Changeable State Variable
Return Next State Value

Fuzzy: MIN

Clicking on each matrix part produces a dialog box for editing that piece of the fuzzy finite state machine. The matrix editing tools include cell and block copying and pasting. Separate commands for adding, subtracting, copying and pasting complete roes(Events) and columns(States) are provided.

Implementation

1. States
2. Events
3. Actions
4. Truth Functions

Implementing and editing the fuzzy state machine is eased through representing it at the top level as a state table. The appearance is the same as a crisp state table except for the explicit addition of the various truth values.

1. The first step as in the crisp is to identify the unique states in which the system will exist. The number of states may be reduced as one state through fuzziness may cover the range of desired values rather than needing a range of states. The initial state must be chosen.
2. Second, identify the system events. These input events may come from outside or may come from internal programmatic action. Again, the fuzzy state machine may require fewer explicit events. These states and events then define the dimensions of a state table. States over the horizontal axis, events over the vertical axis and actions at each of the table matrix cells.
3. Third, define the actions undertaken at each of the table matrix cells. All the steps to this point are the same as a crisp implementation.
4. Create truth functions for each state. This function may be a definition or a calculation based on present State and input Event. Truth value must be provided for the initial State. Subsequent States truths will be defined at state-Transition occurrence.
5. Build this state table representation of the fuzzy state machine as a fuzzy state program function in the language of your choice.

With this fuzzy state program in source or object format the intelligent dynamics of your fuzzy state machine may be included, compiled and executed in your application program.

EXAMPLE 1 (distributed, truths by calculation)

Consider a fuzzy automaton with $E = \{e_1, e_2\}$, $A = \{a_1, a_2, a_3\}$, $S = \{s_1, s_2, s_3, s_4\}$ whose output relations R and state-Transition relation T are defined, respectively, by the matrix and the three-dimensional array

$$\mathbf{R} = \begin{array}{c} \\ s_{P1} \\ s_{P2} \\ s_{P3} \\ s_{P4} \end{array} \begin{array}{c} \begin{array}{ccc} a_1 & a_2 & a_3 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ .5 & 1 & .3 \end{bmatrix} \end{array}, \qquad \mathbf{S} = \begin{bmatrix} \begin{array}{c} e_1 \\ \begin{array}{c|cccc} & s_{N1} & s_{N1} & s_{N1} & s_{N1} \\ \hline s_{P1} & 0 & .4 & .2 & 1 \\ s_{P2} & .3 & 1 & 0 & .2 \\ s_{P3} & .5 & 0 & 0 & 1 \\ s_{P4} & 0 & 0 & 0 & 1 \end{array} \end{array} & \begin{array}{c} e_2 \\ \begin{array}{c|cccc} & s_{N1} & s_{N1} & s_{N1} & s_{N1} \\ \hline s_{P1} & 0 & 0 & 1 & 0 \\ s_{P2} & .2 & 0 & 0 & 1 \\ s_{P3} & 0 & 0 & 0 & 1 \\ s_{P4} & 1 & .3 & 0 & .6 \end{array} \end{array} \end{bmatrix}$$

To describe how this fuzzy automaton operates, let fuzzy sets describing events, actions, and states at any time t be defined by the vectors

$$E^t = [E^t(e_1), E^t(e_2)], \qquad A^t = [A^t(a_1), A^t(a_2), A^t(a_3)],$$

$$S_P^t = [S_P^t(s_1), S_P^t(s_2), S_P^t(s_3), S_P^t(s_4)].$$

Assume now that the initial fuzzy state of the automaton is $S_P^1 = [1 \quad .8 \quad .6 \quad .4]$ and its fuzzy input state is $E^1 = [1 \quad .4]$. Then, using (1) and for example,

$$T_{E^1} = \begin{array}{c|cccc} & s_{N1} & s_{N2} & s_{N3} & s_{N4} \\ \hline s_{P1} & 0 & .4 & .4 & 1 \\ s_{P2} & .3 & 1 & 0 & .4 \\ s_{P3} & .5 & 0 & 0 & 1 \\ s_{P4} & .4 & .3 & 0 & 1 \end{array} .$$

$$\begin{aligned} T_{E^1}(s_1, s_3) &= \max(\min[E^t(e_1), T(e_1, s_1, s_3), \min[E^t(e_2), T(e_2, s_1, s_3)]) \\ &= \max(\min[1, .2], \min[.4, 1]) \\ &= \max(.2, .4) = .4. \end{aligned}$$

To calculate the fuzzy next state E1 and the fuzzy output state B1 of the automaton, we now use (2) and (3):

$$S_N^1 = [1 \quad .8 \quad .6 \quad .4] \circ \begin{bmatrix} 0 & .4 & .4 & 1 \\ .3 & 1 & 0 & .4 \\ .5 & 0 & 0 & 1 \\ .4 & .3 & 0 & 1 \end{bmatrix} = [.5 \quad .8 \quad .4 \quad 1]^{\cdot} \qquad A^1 = [1 \quad .8 \quad .6 \quad .4] \circ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ .5 & 1 & .3 \end{bmatrix} = [1 \quad .8 \quad .6]$$

Assuming now that the next fuzzy input is $\mathbf{A}^2 = [0 \quad 1]$, we obtain

$$S_N^2 = S_N^1 \circ T_{E^2} = [.5 \quad .8 \quad .4 \quad 1] \circ \begin{bmatrix} 0 & 0 & 1 & 0 \\ .2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & .3 & 0 & .6 \end{bmatrix} = [1 \quad .3 \quad .5 \quad .8] \qquad A^2 = S_N^1 \circ \mathbf{R} = [.5 \quad .8 \quad .4 \quad 1] \circ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ .5 & 1 & .3 \end{bmatrix} = [.5 \quad 1 \quad .4]$$

Similarly we can produce larger sequences of fuzzy internal and output states for any given sequence of fuzzy input states.

EXAMPLE 2 (lumped, truths by assignment)

Motor Speed Control

States

Events:	1. Stop, T1 = 1.0	2. Run, T2 = .0
Off, T3 = 1.0	1	1 : T2 = .0
Slower, T4 = 1.0	1	2 : T2 = T2 - .1
Faster, T5 = 1.0	2 : T2 = T2 + .1	2 : T2 = T2 + .1

This 2 by 3 fuzzy state table depict a fuzzy state machine that would require an 11 by 3 crisp state table. The Run truth (T2) takes the place of 10 separate Run states. The first number in each action cell is the next state. The second is an assignment statement on truth value changes. The Run state truth value is the action on motor speed. Exceptions and limits are ignored in this example.

The C fuzzy state program consists of a switch statement on three cases covering the three different action cells. Global arrays keep track of the states and truths. The Event comes from a parameter in the function call.

References and Resources

Papers:

1. Dal Cin , M. [1975a], "Fuzzy state automata: Their stability and fault tolerance." *Intern. J of Computer and Information Sciences*, 4(1), pp. 63-80.
2. Dal Cin , M. [1975b], "Modification tolerance of fuzzy state automata." *Intern. J of Computer and Information Sciences*, 4(1), pp. 81-93.
3. Gaines, B. R. and I. J. Kohout [1975], "The logic of automata." *Intern. J. of General Systems*, 2(4), pp. 191-208.
4. Khan, E. and F. A. Unal [1994], "*Recurrent Fuzzy Logic Using Neural Network.*" The Third IEEE Conference on Fuzzy Systems. pp. 34-40.
5. Khan, E. and F. A. Unal [1994], "*A Fuzzy Finite State Machine Implementation Based on a Neural Fuzzy System.*" 0-7803-1896-X/94 IEEE. 1749-1754.
6. Mizumoto M, J. Toyoda, and K. Tanaka [1969], "Some considerations on fuzzy automata", *J. of Computer and System Sciences*, 3(4), pp. 409-422.
7. Mockof, J. [1991], "A category of fuzzy automata." *Intern. J. of General Systems*, 20(1), pp. 73-82.
8. Ray, A. K., B. Chatterjee and A. K. Majundar. [1991], "A formal power series approach to the construction of minimal fuzzy automata." *Information Sciences*, 55(1-3), pp.189-207.

9. Santos, E. S. [1968], "Maximum automata." *Information and Control*, 13(4), pp. 363-377.
10. Santos, E. S. [1973], "Fuzzy sequential functions." *J. of Cybernetics*, 3(3), pp. 15-31.
11. Santos, E. S. and W. G. Wee [1968], "Realization of fuzzy languages by probabilistic, max-product and maximum automata." *Information Sciences*, 8(1), pp.39-53.
12. Santos, E. S. and W. G. Wee [1968], "General information of sequential machines." *Information and Control*, 8(1), pp.39-53.
13. Wee, W. G. and K. S. Fu [1969], "A formulation of fuzzy automata and its application as a model of learning systems." *IEEE Trans. on Systems, Man and Cybernetics*, 5(3), pp.215-223.

Books:

1. Brubaker, D. I. *Introduction to Fuzzy Logic Systems*. The Huntington Group. Menlo Park, CA 1991.
2. Brubaker, D. I. *Huntington Technical Brief*. The Huntington Group. Menlo Park, CA . March 1993 Number 36.
3. Kandel, A. and C. S. Lee [1979*], Fuzzy Switching and Automata: Theory and Application*. Crane & Russak, New York.
4. Klir, George J. and Yuan, Bo. *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Upper Saddle River, NJ. : Prentice Hall, 1995.
5. Negoita, C. V. and D. A. Ralescu [1975b*], Applications of Fuzzy Sets to Systems Analysis*, Birkhauser, Besel and Stuttgart, and Halsted Press, New York.
6. Pal, S. K. and D. K. D. Majumder [1986], *Fuzzy Mathematical Approach to Pattern Recognition*. John Wiley, New York.
7. Wechler, W. [1978*], The Concept of Fuzziness in Automata and Language Theory.* Academic-Verlag, Berlin.

These writings not only cover theoretical aspects of fuzzy automata, but also explore their applicability in various areas, such as fault-tolerant design, learning, formal languages and grammars, or pattern recognition

Other Books:

1. Edelberg, Allen Y. *Compeditor II C/C++ Finite State Compiler Reference Manual*. AYECO Inc., Orlando, FL
2. McNeill, F. Martin and Thro, Ellen. *Fuzzy Logic: A Practical Approach*. Boston, MA. : Academic Press Professional, 1994.