Conformance relations for fuzzy automata*

Iván Calvo[1], Mercedes G. Merayo[1][0000−0002−4634−4082], Manuel Núñez[1][0000−0001−9808−6401], and Francisco Palomo-Lozano[2][0000−0002−6773−205X]

[1] Departamento Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Spain
{ivcalvo,mgmerayo,manuelnu}@ucm.es
[2] Departmento de Ingeniería Informática, Escuela de Ingeniería
Universidad de Cádiz, Spain
francisco.palomo@uca.es

Abstract. The use of formal methods improves the reliability of computer systems. In this context, fuzzy logic provides a tool to formally specify systems where uncertainty and imprecision play an important role. In this paper, we propose an extension of the fuzzy automata formalism and establish different conformance relations. The main goal of these relations is to formally capture the idea of a system behaving as specified by a specification. We sketch how our conformance relations can be alternatively characterized as a testing process by producing sound and complete sets of tests.

Keywords: Fuzzy automata · Conformance relations · Formal approaches to testing.

1 Introduction

The use of formal methods can notoriously improve the reliability of systems: a formal development, with a mathematical basis, will help to skip faults. In particular, the existence of a formal specification will guide the development of the systems and will help to assess the correctness of the developed system. Unfortunately, Computer Science is a *strange* Engineering discipline because it is not usually understood, specially in industrial environments, the importance of a formal plan to build complex systems [9]. The second mentioned use of a specification, validate the developed system, has been combined with testing [1] to provide a powerful methodology: formal approaches to testing [6, 8]. Essentially, formal testing takes a specification (the formal description of the system that we are developing) and a system under test (SUT, the implementation of the system) and checks, by applying tests, whether the SUT shows a behavior contradicting what the specification states. There are many alternative formal testing frameworks but the ones considering state-based systems are more frequent and are usually supported by tools [12].

* Research partially supported by the Spanish MINECO/FEDER project DArDOS (TIN2015-65845-C3-1-R and TIN2015-65845-C3-3-R) and the Comunidad de Madrid project FORTE-CM (S2018/TCS-4314).

The main problem with mainstream formalisms is that they are not suited to specify systems with specific features. These formalisms are appropriate to specify properties such as "if the system receives input i then it should produce output o", that is, causality relations that should always hold. However, there are systems where this information can be more imprecise. For example, there are situations where we would like to specify a property such as "if the system receives an input i then it should produce output o most of the times, while it should produce output o' sometimes". In this case, we need a *tailored* formalism. In this paper we will consider fuzzy logic [15, 16] as the theory underlying our *fuzzy automata*. There are many proposals to include fuzzy logic into automata [7, 11, 14] and our research group has been particularly active in this area [2–5]. The main goal of this paper is to present a formal framework to assess the correctness of an SUT with respect to a specification expressed as a fuzzy automata. The first step was to take our last formalism [4] and slightly modify it to incorporate the improvements that we have detected since its creation. Next, we had to provide an operational semantics to formally define how a system evolves. This is a fundamental step because conformance relations rely on comparing the behavior of the specification and the SUT. The study of an appropriate conformance relation revealed that in a *fuzzy framework* there is not a unique way to define it. Therefore, we propose alternative notions and show how they are related. A final step is to characterize conformance relations via testing. Due to space limitations we cannot present the complete testing framework and we only sketch how tests are defined and how test derivation is implemented.

The rest of the paper is structured as follows. In Section 2 we review some concepts related to fuzzy logic. In Section 3 we introduce our latest proposal of fuzzy automata by formally defining its syntax and semantics. In this section we also introduce some relevant concepts about the operational behavior of these automata. In Section 4 we define our conformance relations and include some interesting observations and results, which will be needed, in Section 5, to define our approach to testing. Finally, in Section 6 we present our conclusions and some lines for future work.

2 Preliminaries

In this section we review the basic definitions underlying our formalism. These definitions represent an extension of the concepts presented in our previous work [4] but revised and refined in order to fit the needs of the current version of our formalism. In usual *crisp* logic we have that truth values can be either *True* or *False*, represented respectively by 1 and 0. In contrast, *fuzzy* logic allows truth values to be anywhere in between these bounds. In order to combine these truth values, we need to define the concept of *triangular norms* or t-norms.

Definition 1. *A function* $\triangle : [0,1] \times [0,1] \longrightarrow [0,1]$ *is said to be a* t*-norm when it satisfies the following properties:*

- $\triangle$ *is commutative:* $x \triangle y = y \triangle x$ *for all* $x, y \in [0,1]$.
- $\triangle$ *is associative:* $(x \triangle y) \triangle z = x \triangle (y \triangle z)$ *for all* $x, y, z \in [0,1]$.
- $\triangle$ *is monotonic: if* $x_1 \leq y_1$ *and* $x_2 \leq y_2$ *then* $x_1 \triangle x_2 \leq y_1 \triangle y_2$ *for all* $x_1, x_2, y_1, y_2 \in [0,1]$.

– $\triangle$ *has an identity element:* $1 \triangle a = a$ *for all* $a \in [0, 1]$.

In this paper we will only consider strictly monotonic t*-norms, that is,* t*-norms such that for all* $0 \leq x, y, z \leq 1$ *we have that* $y < z$ *and* $x > 0$ *implies* $(x \triangle y) < (x \triangle z)$.

Some examples of strictly monotonic t-norms are the Hamacher t-norm

$$(x, y) \mapsto \frac{x \cdot y}{x + y - x \cdot y}$$

and the product t-norm

$$(x, y) \mapsto x \cdot y$$

Not strictly monotonic t-norms do not have the properties that we need and are not considered in this paper. For example, the minimum t-norm

$$(x, y) \mapsto \min(x, y)$$

is not strictly monotonic. Our proposed formalism is based on finite automata equipped with a finite set of real valued variables. These variables are used to track and process relevant data. A precise definition of the syntax and semantics of arithmetic expressions over the set of variables needs to be defined in order to specify how data is meant to be processed.

Definition 2. Expressions *are formed by variable evaluations, usual arithmetic operators* $(+, -, \cdot, /)$*, clamped* t*-norms (functions that are* t*-norms on the interval* $[0, 1]$ *and identical to* 0 *outside the interval), and the reserved variable* μ*.*

The alphabet labelling automata transitions includes *input* and *output actions*.

Definition 3. *An* input action *consists of its name, followed by a tuple of variable names taking real values. The name is a string whose first character is* ?*. The set of all input actions is denoted by* I*. An* output action *consists of its name, followed by a tuple of real-valued expressions. The name is a string whose first character is* !*. The set of all output actions is denoted by* O*.*

We assume that all the actions have different names, so that there is no pair of actions with the same name and different associated tuples. The set of all defined actions is denoted by $\mathrm{Acts} = I \bigcup O$*.*

The actions received and produced by a system are parameterized with actual values instead of variable names or expressions.

Definition 4. *Let* Acts *be a set of actions. We denote by* IActs *the set of actions from* Acts *parametrized with all possible tuples of values of the same arity as the corresponding tuple of variable names or expressions.*

For example, consider an input $?i(x) \in \mathrm{Acts}$. Then we have that $?i(r)$ belongs to IActs for all $r \in \mathbb{R}$. Similarly, if an output $!o(x + 1) \in \mathrm{Acts}$ then $!o(r) \in \mathrm{IActs}$ for all $r \in \mathbb{R}$.

A test case is determined by a sequence of input actions and output names. The idea is that we will apply inputs with specific values to an SUT and receive outputs from it. The elements forming these sequences will be called *test actions*.

Definition 5. *Let* $\mathrm{Acts} = I \cup O$ *be a set of actions. We define an input action for each input in* I *and an output action for each output in* O. Test inputs *consist of their name and a tuple of values.* Test outputs *consist only of their name. The set of all test inputs and outputs is called* TActs.

We will also define a mapping from IActs to TActs which removes output parameters. This will be used in the definition of our conformance relations.

Definition 6. *Let* Acts *be a set of actions,* IActs *be its associated set of parameterized actions and* TActs *be the set of test actions from* Acts. *The mapping* $\mathcal{D} : \mathrm{IActs} \longrightarrow \mathrm{TActs}$ *is defined as:*

$$\mathcal{D}(a) = \begin{cases} ?i(\chi) & \text{if } a = ?i(\chi) \\ !o & \text{if } a = !o(\zeta) \end{cases}$$

Given a sequence $a_1 \cdots a_n \in \mathrm{IActs}^*$ *we define* $\mathcal{D}(a_1, \cdots, a_n)$ *in the natural way, that is as* $\mathcal{D}(a_1) \cdots \mathcal{D}(a_n) \in \mathrm{TActs}^*$.

Variables are updated when a transition of an automaton is triggered. In order to include this idea, each transition is labelled with a *variable transformation*.

Definition 7. *Let* $x_1, \ldots, x_m$ *be variable names and* $t_1, \ldots, t_m$ *be real valued expressions. A* variables transformation*, denoted by* $[t_1/x_1, \ldots, t_m/x_m]$*, assigns to each variable* x_i *the value obtained after evaluating the term* t_i*. The evaluations of all the terms are computed before any assignment is performed. The set of variable transformations is denoted by* $\mathcal{VT}$.

We use *fuzzy relations*, instead of the usual *crisp* relations (e.g. $\leq, \geq, =$), to compare input values. The fuzzy counterparts are functions from $\mathbb{R}^2$ to $[0, 1]$, taking the value 1 when the corresponding crisp relation holds. For example, the fuzzy relation $\overline{x \leq y}^{\delta}$, defined as

$$\overline{x \leq y}^{\delta} \equiv \begin{cases} 1 & \text{if } x < y \\ \frac{\delta + y - x}{\delta} & \text{if } y \leq x \leq y + \delta \\ 0 & \text{if } y + \delta < x \end{cases}$$

represents an extension of the crisp relation $x \leq y$. An extended discussion on fuzzy relations can be found in previous work [3].

Transitions may be triggered depending on the parameters of the received input action. In addition, a transition will be triggered with a certain *Grade of Confidence* belonging to the interval $[0, 1]$. More specifically, a transition will be triggered with a Grade of Confidence equal to the *Satisfaction Degree* of its associated fuzzy constraint.

Definition 8. *A fuzzy constraint is a formula consisting of fuzzy relations, possibly combined with t-norms, which may have free variables. The set of all fuzzy constraints is denoted by* $\mathcal{FC}$*. When the set of free variables of a constraint,* C*, is contained in the tuple of variable names of an input action,* $?i(x_1 \cdots x_n)$*, then we can define the function* $\mu_{C,?i(x_1 \cdots x_n)}(v_1 \cdots v_n)$ *as the value obtained by evaluating the constraint* C *after performing the substitution* $x_1 \mapsto v_1, \ldots, x_n \mapsto v_n$*. This value represents the Satisfaction Degree of the constraint with respect to the input values* $v_1, \ldots, v_n$.

3 Fuzzy automata: syntax and semantics

In this section we review the basic syntax and semantics of our formalism and extend it with alternative semantics. These alternative semantics will be used in the definition of the proposed conformance relations. First, we introduce our notion of fuzzy automata.

Definition 9. *A* fuzzy automaton *is a tuple*

$$(S, \mathrm{Acts}, L, X, X_0, s_0, Tr)$$

where:

- S *is a finite set of states.*
- Acts *is a finite set of actions, partitioned into a set of inputs* I *and a set of outputs* O*.*
- $L \subseteq O$ *is a distinguished subset of outputs. We refer to these actions as localized outputs.*
- X *is a set of variable names.*
- $X_0 : X \longrightarrow \mathbb{R}$ *is the initial mapping from variable names to real values. We call these mappings variable states.*
- $s_0 \in S$ *is the initial state.*
- $Tr \subseteq S \times \mathrm{Acts} \times \mathcal{FC} \times \mathcal{VT} \times S$ *is a set of transitions satisfying the following conditions:*
  - *Output transitions have the constant True as constraint.*
  - *Input transitions have a constraint whose free variables are a subset of the variables of the input action.*
  - *For each localized output,* $l \in L$*, there exists a state,* $s \in S$*, such that, for all transitions of the form* $(s_1, l, \mathrm{True}, \mathrm{vt}, s_2) \in Tr$*, we have* $s_2 = s$*.*

Now, we will review the basic semantics of our version of *fuzzy automata*.

Definition 10. *Let* $A = (S, \mathrm{Acts}, L, X, X_0, s_0, Tr)$ *be a fuzzy automaton and* $\triangle$ *be a strictly monotonic* t*-norm. Let* $s_1, s_2 \in S$ *be states and* $X_1, X_2 : X \longrightarrow \mathbb{R}$ *be variable states. We have a* transition *from* (s_1, X_1) *to* (s_2, X_2)*, after performing the* input action $?i(\chi) \in I$ *for the real valued tuple* α *with confidence* ϵ*, denoted by*

$$(s_1, X_1) \xrightarrow{?i(\alpha)}_{\epsilon} (s_2, X_2)$$

if the following conditions hold:

- *There exists a fuzzy constraint* $C \in \mathcal{FC}$ *such that* $(s_1, ?i(\chi), C, VT, s_2) \in Tr$*.*
- *The grade of confidence of* C*, with the corresponding variables mapped to the values in* α*, is higher than* 0*. We denote this grade of confidence by* $\epsilon := \mu_{C,?i(\chi)}(\alpha)$*.*
- X_2 *is the result of applying* VT *to* X_1 *considering:*
  - *The reserved variable* μ *takes the value* ϵ*.*
  - *The values of the variables in the expressions of* VT *are taken from* α *when the corresponding variable belongs to the input* $?i(\chi)$*. The values for the rest of variables in the expressions of* VT *are taken from* X_1*.*

We have a transition *from* (s_1, X_1) *to* (s_2, X_2)*, after performing the* output action $!o(\zeta) \in O$*, denoted by*

$$(s_1, X_1) \xrightarrow{!o(\alpha)}_1 (s_2, X_2)$$

if the following conditions hold:

- *There exists* $(s_1, !o(\zeta), True, VT, s_2) \in Tr$.
- X_2 *is the result of applying* VT *to* X_1 *and* α *is the tuple of expressions from the output* o*, considering:*
  - *The reserved variable* μ *takes the value 1.*
  - *The values of the variables in the expressions,* ζ*, are taken from* X_1.

We say that a sequence

$$(s_0, X_0) \xrightarrow{a_1(\alpha_1)}_{\epsilon_1} (s_1, X_1) \cdots \xrightarrow{a_n(\alpha_n)}_{\epsilon_n} (s_n, X_n)$$

of consecutive transitions starting in the initial state of the automaton is a $\triangle$-trace *of* A *if* $\epsilon = \triangle\{\epsilon_1, \ldots, \epsilon_n\}$ *is greater than zero. In this case, we write*

$$(s_0, X_0) \xRightarrow{a_1(\alpha_1)\cdots a_n(\alpha_n)}_{\epsilon} (s_n, X_n)$$

Next we will define two transition concatenation rules that will be later used to define the proposed conformance relations. The first one is based upon the idea of imposing a minimum confidence level.

Definition 11. *Let* $A = (S, \mathrm{Acts}, L, X, X_0, s_0, Tr)$ *be a fuzzy automaton,* $\alpha \in [0, 1]$ *and* $a_1 \cdots a_n \in \mathrm{IActs}^+$ *be a sequence of actions. We write*

$$(s_0, X_0) \xRightarrow{a_1 \cdots a_n}_{\epsilon}{}^{\mathrm{conf}^\alpha} (s, X)$$

if $\epsilon > \alpha$ *and we have* $(s_0, X_0) \xRightarrow{a_1 \cdots a_n}_{\epsilon} (s, X)$.

The difference with the basic transition semantics is that conf^α does not consider transitions below a certain confidence threshold. The following result means that the higher α is, the more restrictive conf^α becomes (the proof is immediate).

Proposition 1. *Let* $A = (S, \mathrm{Acts}, L, X, X_0, s_0, Tr)$ *be a fuzzy automaton,* $a_1 \cdots a_n \in \mathrm{IActs}^+$ *be a sequence of actions and* $\alpha_1, \alpha_2 \in [0, 1]$ *such that* $\alpha_1 < \alpha_2$*. We have that*

$$(s_0, X_0) \xRightarrow{a_1 \cdots a_n}_{\epsilon}{}^{\mathrm{conf}^{\alpha_2}} (s, X) \text{ implies } (s_0, X_0) \xRightarrow{a_1 \cdots a_n}_{\epsilon}{}^{\mathrm{conf}^{\alpha_1}} (s, X)$$

The second transition concatenation rule is a little more complex. Its purpose is to accept only the most likely executions.

Definition 12. *Let* $A = (S, \mathrm{Acts}, L, X, X_0, s_0, Tr)$ *be a fuzzy automaton and* $a_1 \cdots a_n \in \mathrm{IActs}^+$ *be a sequence of actions. We write*

$$(s_0, X_0) \xRightarrow{a_1 \cdots a_n}_{\epsilon}{}^{\mathrm{confm}} (s, X)$$

if we have that

$$(s_0, X_0) \xRightarrow{a_1 \cdots a_n}_{\epsilon} (s, X)$$

and one of the following four conditions holds.

- $n = 1$ *and* $\mathcal{D}(a_1) \notin L$.
- $n = 1$, $\mathcal{D}(a_1) \in L$ *and for all* a'_1 *such that* $\mathcal{D}(a_1) = \mathcal{D}(a'_1)$ *and such that* $(s_0, X_0) \xRightarrow{a'_1}_{\epsilon'} (s, X)$ *we have* $\epsilon \geq \epsilon'$.
- $n > 1$, $\mathcal{D}(a_n) \notin L$ *and there exist* $\epsilon', \epsilon'', s'$ *and* X' *such that* $\epsilon = \epsilon' \triangle \epsilon''$, $(s_0, X_0) \xRightarrow{a_1 \cdots a_{n-1}}{}^{\text{confm}}_{\epsilon'} (s', X')$ *and* $(s', X') \xrightarrow{a_n}_{\epsilon''} (s, X)$.
- $n > 1$, $\mathcal{D}(a_n) \in L$ *and for all* $a'_1 \cdots a'_n \in \text{IActs}^*$ *such that* $\mathcal{D}(a_1 \cdots a_n) = \mathcal{D}(a'_1 \cdots a'_n)$ *and* $(S_0, X_0) \xRightarrow{a'_1 \cdots a'_n}_{\epsilon'} (S, X')$ *we have that* $\epsilon \geq \epsilon'$.

The idea behind this definition is that if an action in L is received, then the system must reach a specific state: the state corresponding to that particular action in L. Having this property in mind, we know that the higher the Grade of Confidence is at that point, the higher it will be after successive actions. Therefore, the previous definition translates into imposing that output actions must be parameterized with values that maximize the Grade of Confidence of that execution.

4 Conformance relations

In this section we define our conformance relations. The general framework considers a *System Under Test (SUT)*, which is receiving input actions and producing output actions. We will say that an SUT conforms to a specification if every sequence of actions observed in the SUT does not show a mismatch with respect to what the specification says. In addition, we need to parameterize the conformance relations with respect to a specific t-norm. Formally, our two relations should have a $\triangle$ parameter, that is, a strictly monotonic t-norm. However, we will omit this relation to not overload the notation. The definition of our conformance relations is based on the operational semantics previously defined in Section 3. We first define a conformance relation based on accepting the sequences with the highest Grade of Confidence.

Definition 13. *Let* $Spec = (S, \text{Acts}, L, X, X_0, s_0, Tr)$ *be a fuzzy automaton. We say that an SUT maximally conforms to Spec, and we write*

$$\text{SUT} \ \texttt{confmax} \ Spec$$

if for all sequence of actions $a_1 \cdots a_n \in \text{IActs}^*$ *observed in the SUT, we have that there exist* ϵ, s *and* X *such that* $(s_0, X_0) \xRightarrow{a_1 \cdots a_n}{}^{\text{confm}}_{\epsilon} (s, X)$.

The `confmax` relation only holds when every observed sequence of actions is *accepted* by the specification. In particular, sequences that have not terminated yet should also be *accepted* if the complete sequence is going to be *accepted*. This property is satisfied because of the following property of our operational semantics. The proof is based on the strict monotonicity of the t-norms that we consider in our framework.

Proposition 2. *Let* $A = (S, \text{Acts}, L, X, X_0, s_0, Tr)$ *be a fuzzy automaton and* $a_1 \cdots a_n \in \text{IActs}^*$ *be a sequence of actions. If we have*

$$(s_0, X_0) \xRightarrow{a_1, \cdots, a_n}{}^{\text{confm}}_{\epsilon} (s, X)$$

for some state s, confidence ϵ and variable state X, then we also have that there exist ϵ', S' and X' such that

$$(s_0, X_0) \stackrel{a_1,\cdots,a_{n-1}}{\Longrightarrow}_{\epsilon'}^{\text{confm}} (s', X')$$

Next, we define a conformance relation based on accepting only sequences whose *Grade of Confidence* is higher than a certain bound.

Definition 14. *Let $Spec = (S, \text{Acts}, L, X, X_0, s_0, Tr)$ be a fuzzy automaton and $\alpha \in [0, 1]$. We say that an SUT conforms to Spec with respect to α, and we write*

$$\text{SUT } \texttt{conf}^{\alpha} \ Spec$$

if for all sequence of actions $a_1 \cdots a_n \in \text{IActs}^*$ observed in the SUT, we have that there exist ϵ, S and X such that

$$(s_0, X_0) \stackrel{a_1,\cdots,a_n}{\Longrightarrow}_{\epsilon}^{\text{conf}^{\alpha}} (s, X)$$

Similarly to the previous case, given an SUT that conforms to a specification with respect to the $\texttt{conf}^{\alpha}$ relation, if a sequence of actions is observed in the SUT then every prefix of the sequence has been observed too. The proof of this result relies again in the monotonicity of t-norms.

Proposition 3. *Let $A = (S, \text{Acts}, L, X, X_0, s_0, Tr)$ be a fuzzy automaton and $a_1 \cdots a_n \in \text{IActs}^*$ be a sequence of actions. If we have*

$$(s_0, X_0) \stackrel{a_1\cdots a_n}{\Longrightarrow}_{\epsilon}^{\text{conf}^{\alpha}} (s, X)$$

for some state s, confidence ϵ and variable state X, then we also have that there exist ϵ', S' and X' such that

$$(s_0, X_0) \stackrel{a_1\cdots a_{n-1}}{\Longrightarrow}_{\epsilon'}^{\text{conf}^{\alpha}} (s', X')$$

Finally, we define a conformance relation that imposes both a certain bound and the maximal *Grade of Confidence* condition.

Definition 15. *Let $Spec = (S, \text{Acts}, L, X, X_0, s_0, Tr)$ be a fuzzy automaton and $\alpha \in [0, 1]$. We say that an SUT maximally conforms to Spec with respect to α, and we write*

$$\text{SUT } \texttt{confmax}^{\alpha} \ Spec$$

if for all sequence of actions $a_1 \cdots a_n \in \text{IActs}^*$ observed in the SUT we have that there exist ϵ, S, and X such that

$$(s_0, X_0) \stackrel{a_1\cdots a_n}{\Longrightarrow}_{\epsilon}^{\text{conf}^{\alpha}} (s, X) \text{ and } (s_0, X_0) \stackrel{a_1\cdots a_n}{\Longrightarrow}_{\epsilon}^{\text{confm}} (s, X)$$

This relation represents the conjunction of the former two relations, as stated in the following result.

Proposition 4. *If we have* $\text{SUT } \texttt{confmax}^{\alpha}$ *Spec then we also have* $\text{SUT } \texttt{confmax}$ *Spec and* $\text{SUT } \texttt{conf}^{\alpha_1}$ *Spec, for all $\alpha_1 \leq \alpha$.*

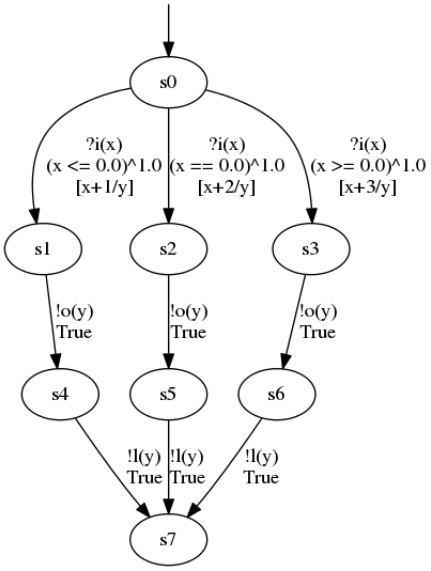

Fig. 1. Example of a fuzzy automaton.

5 Test definition and generation

Although conformance relations give a precise notion to define when an SUT conforms to a specification, in other words, when we have no evidence to claim that the system under development is incorrect, it is important to have an alternative characterization of these relations. Testing provides such a tool. The idea is to define a testing framework such that the SUT successfully passes a set of tests (extracted from the specification) if and only if the conformance of the SUT with respect to the specification holds. Due to space restrictions we cannot present all the details. We will follow the classical approach where conformance was characterized as a testing framework [13] and we will focus on the non-standard intricacies that we had to confront. During the rest of this section we will briefly describe the data structure that will represent each test and we will define what does it mean for an SUT to pass a test. We will also sketch a test derivation algorithm that produces sound and complete test suites with respect to a given specification and a conformance relation.

We begin by defining the trees that represent the possible executions of a particular test case.

Definition 16. *A test tree is defined to be a rooted tree whose edges are labelled with actions from* IActs*. The set of all test trees is denoted by* TTrees*.*

An example of a *test tree* is shown in Figure 2. This test tree corresponds to a particular sequence of test actions. Next, we have to define what it means for a sequence of actions to satisfy a given test tree.

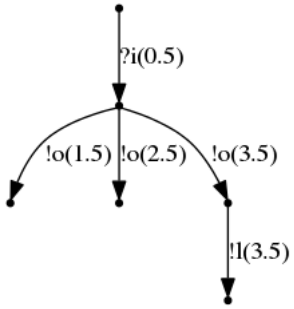

Fig. 2. Test tree corresponding to the test action sequence $?i(0.5), !o, !l$ for the automaton given in Figure 1.

Definition 17. *We say that a sequence* $a_1 \cdots a_n \in \mathrm{IActs}^*$ *satisfies a test tree* $T \in \mathrm{TTrees}$ *if there exists a sequence of adjacent edges in* T*, starting in its root, labelled consecutively with the actions forming* $a_1 \cdots a_n$*. In this case we write* $a_1 \cdots a_n \vDash T$*; otherwise, we write* $a_1 \cdots a_n \nvDash T$

The idea is that a test tree provides a compact representation of the set of accepted actions sequences. We can now define what does it mean to pass or fail a given test case.

Definition 18. *A test case is defined to be a pair* (t, T) *with* $t \in \mathrm{TActs}^*$ *and* $T \in \mathrm{TTrees}$*. A sequence of actions* $a_1 \cdots a_n \in \mathrm{IActs}^*$ *observed in the SUT fails the test* (t, T)*, and we write* $a_1 \cdots a_n \nvDash (t, T)$*, if* $\mathcal{D}(a_1 \cdots a_n)$ *is a prefix of* t *and* $a_1 \cdots a_n \nvDash T$*; otherwise we say that the test is passed and we write* $a_1 \cdots a_n \vDash (t, T)$

Intuitively, sequences of test actions are associated with test trees in the sense that if a sequence of actions is a fragment of the sequence of test actions, then it must be a fragment of the corresponding test tree. The most important property that a test must have is to be *sound*. This means that the test only fails when the sequence of actions is not valid with respect to a specification. In order to determine whether a sequence of actions is valid, both a specification and a conformance relation must be considered.

Definition 19. *Let* $Spec = (S, \mathrm{Acts}, L, X, X_0, s_0, Tr)$ *be a fuzzy automaton. A test case* $(t, T) \in \mathrm{TActs}^* \times \mathrm{TTrees}$ *is said to be sound with respect to* $Spec$ *under the conformance relation* `rel` *if for each sequence of actions* $a_1 \cdots a_n$ *we have that*

$$(s_0, X_0) \stackrel{a_1 \cdots a_n}{\Longrightarrow}_{\epsilon}^{\mathrm{rel}} (s, X)$$

implies that $a_1 \cdots a_n \vDash (t, T)$.

For example, in Figure 2 we show a sound test case with respect to the specification given in Figure 1 and the conformance relation `confmax`. We can see it is sound since after receiving $?i(0.5)$ the only possible outputs are $!o(1.5)$, $!o(2.5)$ and $!o(3.5)$ and,

after that, the localized action $!l$ must take the value $!l(3.5)$ since it is the only value which maximices the Grade of Confidence in the sequence of actions.

The correctness of an SUT is tested by applying a *test suite*.

Definition 20. *A test suite is defined to be a set of tests. An SUT is said to pass a test suite if for each sequence of actions observed during the application of all the tests to the SUT we have that the tests are passed.*

The most important property that a test suite may have, once we know that all the tests are sound, is to be *complete*. This means that if the SUT shows a sequence of actions that it is not valid with respect to a specification, then at least one of the tests will fail. In order for a test suite to be actually complete it would need to contain both an infinite number of parameters for each action and possibly also an infinite number of actions. In practice, samples from random variables modelling the distribution of the parameters may be used in order to produce a realistic pool of parameterized input actions. A longer explanation about this (classical) problem to achieve real completeness can be found in previous work [10].

The last step of our framework was to define a *test derivation algorithm* (the actual algorithm cannot be included in the paper due, as we have already said, to space limitations). Essentially, we traverse the specification and generate tests that appropriately reflect the structure of the specification (the interested reader can see similar algorithms, although for simpler formalisms, in previous work [10, 13]).

6 Conclusions and future work

Designing and building software with the aid of formal methods and models is certainly one of the most promising ways to improve the reliability of critical systems. Fuzzy logic has proven to offer a powerful and expressive language to model and reason about the implicit uncertainty present in many fields.

In this paper we have successfully extended our previous framework based on fuzzy automata, making it more suitable to specify a wide variety of software systems. A family of conformance relations has been defined, taking advantage of the monotonicity properties of t-norms, which makes them consistent with respect to the partial execution of an implementation. This translates into a better suitability to model systems meant to run over long periods of time, whose execution at some point before completion may need to be proved correct. A testing methodology has been presented in order to apply these formal notions to actual software.

Our next goal consists in applying this extension to real use cases in order to show the potential of the proposed framework. Besides developing models relying on the framework, software tools are planned to be constructed in order to automatize as much as possible the generation of test cases. This task implies both the need to formally prove the soundness of the different test generation algorithms and to empirically show the adequacy of different strategies meant to choose a rich enough subset of the sound test cases in order to obtain complete test suites.

References

1. P. Ammann and J. Offutt. *Introduction to Software Testing*. Cambridge University Press, 2nd edition, 2017.
2. C. Andrés, L. Llana, and M. Núñez. Self-adaptive fuzzy-timed systems. In *13th IEEE Congress on Evolutionary Computation, CEC'11*, pages 115–122. IEEE Computer Society, 2011.
3. J. Boubeta-Puig, A. Camacho, L. Llana, and M. Núñez. A formal framework to specify and test systems with fuzzy-time information. In *14th Int. Work-Conf. on Artificial Neural Networks, IWANN'17, LNCS 10306*, pages 403–414. Springer, 2017.
4. I. Calvo, M. G. Merayo, and M. Núñez. An improved and tool-supported fuzzy automata framework to analyze heart data. In *10th Asian Conference on Intelligent Information and Database Systems, ACIIDS'18, LNAI 10751*, pages 694–704. Springer, 2018.
5. A. Camacho, M. G. Merayo, and M. Núñez. Using fuzzy automata to diagnose and predict heart problems. In *19th IEEE Congress on Evolutionary Computation, CEC'17*, pages 846–853. IEEE Computer Society, 2017.
6. A. R. Cavalli, T. Higashino, and M. Núñez. A survey on formal active and passive testing with applications to the cloud. *Annales of Telecommunications*, 70(3-4):85–93, 2015.
7. M. Doostfatemeh and S. C. Kremer. New directions in fuzzy automata. *International Journal of Approximate Reasoning*, 38(2):175–214, 2005.
8. R. M. Hierons, K. Bogdanov, J.P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Luettgen, A.J.H Simons, S. Vilkomir, M.R. Woodward, and H. Zedan. Using formal specifications to support testing. *ACM Computing Surveys*, 41(2):9:1–9:76, 2009.
9. L. Lamport. Who builds a house without drawing blueprints? *Communications of the ACM*, 58(4):38–41, 2015.
10. M. G. Merayo, M. Núñez, and I. Rodríguez. Formal testing from timed finite state machines. *Computer Networks*, 52(2):432–460, 2008.
11. J. N. Mordeson and D. S. Malik. *Fuzzy automata and languages: theory and applications*. Chapman & Hall/CRC, 2002.
12. M. Shafique and Y. Labiche. A systematic review of state-based test tools. *International Journal on Software Tools for Technology Transfer*, 17(1):59–76, 2015.
13. J. Tretmans. Model based testing with labelled transition systems. In *Formal Methods and Testing, LNCS 4949*, pages 1–38. Springer, 2008.
14. W. G. Wee and K. S. Fu. A formulation of fuzzy automata and its application as a model of learning systems. *IEEE Transactions on Systems Science and Cybernetics*, 5(3):215–223, 1969.
15. L. A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.
16. L. A. Zadeh. *Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems*. Advances in Fuzzy Systems - Applications and Theory: vol. 6. World Scientific Press, 1996.