# Alloy Goes Fuzzy

Pedro Silva$^{1,3[0000-0001-6918-5558]}$, Alcino Cunha$^{1,3[0000-0002-2714-8027]}$, Nuno Macedo$^{2,3[0000-0002-4817-948X]}$, and José N. Oliveira$^{1,3[0000-0002-0196-4229]}$

[1] Universidade do Minho, Braga, Portugal {alcino,jno}@di.uminho.pt
[2] Universidade do Porto, Porto, Portugal nmacedo@fe.up.pt
[3] INESC TEC, Porto, Portugal pedro.d.silva@inesctec.pt

**Abstract.** Humans are good at understanding subjective or vague statements which, however, are hard to express in classical logic. Fuzzy logic is an evolution of classical logic that can cope with vague terms by handling *degrees of truth* and not just the crisp values true and false.
Logic is the formal basis of computing, enabling the formal design of systems supported by tools such as model checkers and theorem provers. This paper shows how a model checker such as Alloy can evolve to handle both classical and fuzzy logic, enabling the specification of high-level quantitative relational models in the fuzzy domain.
In particular, the paper showcases how QAlloy-F (a conservative, general-purpose quantitative extension to standard Alloy) can be used to tackle fuzzy problems, namely in the context of validating the design of fuzzy controllers. The evaluation of QAlloy-F against examples taken from various classes of fuzzy case studies shows the approach to be feasible.

**Keywords:** Fuzzy relations · Formal methods · Model checking · Alloy.

## 1 Introduction

Expressing subjective statements based on personal experience is something that comes naturally to people. Humans hear remarks such as “the weather is cold today” and have no problem making sense of the meaning. However, when communicating with machines, conveying such phrases becomes challenging, as machines are used to dealing with precise information only. After all, from the previous statement alone one is just unable to determine what the exact outside temperature is. In fact, that very same phrase may be said under drastically different values of the temperature, depending on who is saying it, their background, where they are located and the climate that they are used to.

Zadeh [39] establishes a means to express such statements through *fuzzy set theory* (more commonly referred to as *fuzzy logic*), arising as a generalization of classical set theory. Rather than making use of Boolean values, one reasons over real numbers which, within the unit interval, describe *degrees of truth.* Since Zadeh’s pioneering work there has been a lot of interest in fuzzy logic and fuzzy systems, both from the theoretical and practical sides, see e.g. [9,28,38,17]. Moreover, tools such as MATLAB®’s Fuzzy Logic Toolbox™ [14] and FuzzyLite [27] provide means of designing fuzzy controllers, exploring different inputs, experimenting with different values of the inputs and seeing the resulting behaviour.

However, as far as the authors are aware, there are no tools that provide high-level modelling and verification techniques readily available for structural modelling within the fuzzy domain. Existing work either acts at a low-level of abstraction [3,37,36], or is concerned with checking the behaviour of fuzzy systems [26,31,23,35]. We argue that, precisely due to the uncertain nature of fuzzy systems, high-level tools that can be used by domain experts are essential, whereby alternative designs can be explored in the early stages of development.

Alloy [13] excels in such structural modelling when standard logic is used, and a recent extension by this team, QAlloy [30], allows reasoning about quantitative models. The main contribution of this paper is to showcase how a variant of QAlloy can be used to tackle fuzzy problems through fuzzy relational reasoning, thus enabling one to specify high-level quantitative relational models in the fuzzy domain. We denote this variant as QAlloy-F, and consider it orthogonal to the originally proposed QAlloy-I for the integer domain.

The rest of the paper is structured as follows: Section 2 provides some background on fuzzy logic. Section 3 presents QAlloy-F by example, followed by its formal presentation in Section 4. Section 5 shows its application to the design of fuzzy controllers and Section 6 evaluates the tool. Lastly, Section 7 presents related work and Section 8 wraps up the paper.

## 2 Background

Standard predicate calculus deals with either true or false statements. It is common to represent the truth values *false* by 0 and *true* by 1 subject to logic operators, making a Boolean algebra. For instance, given $p, q \in \{0, 1\}$ the multiplication $p \cdot q$ expresses logic *conjunction*, $p \wedge q$. Similarly, $p \vee q = p + q - p \cdot q$ defines logic *disjunction*, $\neg p = 1 - p$ logic *negation*, and so on.

The use above of simple arithmetic operations over $\{0, 1\}$ to express Boolean operators is suggestive of a possible generalization: what if one uses the *whole interval* $[0, 1]$ of real numbers and not just its boundaries $\{0, 1\}$? The closer some $p \in [0, 1]$ would be to 1 (resp. 0), the "more true" (resp. "more false") it would be. The arithmetic operator definitions could stay the same and one would deal with *degrees of truth* rather than discrete truth or falsehood, obtaining a more expressive and flexible logical framework. Such is the motivation and formal basis of *fuzzy logic* [39].

*Fuzzy logic.* As hinted above, in fuzzy logic the standard logic connectives $x \wedge y$ and $x \vee y$ become real number functions of type $[0, 1] \times [0, 1] \to [0, 1]$. Instead of $x \wedge y$, the notation $t(x, y)$ is used in fuzzy logic, $t$ being named the *triangular norm* (or *t-norm*, for short). Dually, disjunction is expressed by $s(x, y)$, the so-called *triangular conorm* (abbrev. *t-conorm*). To make formulæ more readable we adopt the infix notations $x \sqcap y$ for $t(x, y)$ and $x \sqcup y$ for $s(x, y)$. Triangular (co)norms may vary in fuzzy logic. Table 1 gives a few standard definitions, the so-called *algebraic product* being the instance that opened this section. In general, the t-norm is a binary operation that must enjoy the properties of commutativity, associativity, monotonicity and respect the boundary condition $x \sqcap 1 = x$ for any $x \in [0, 1]$. The dual t-conorm must abide to the same properties, its boundary condition being $0 \sqcup x = x$, for all $x \in [0, 1]$. The algebraic product, Gödelian and Łukasiewicz t-norms of Table 1 are the most popular.

**Table 1.** Triangular norms and conorms

| **t-norm** | $x \sqcap y$ | **t-conorm** | $x \sqcup y$ |
|---|---|---|---|
| Gödelian minimum | $\min(x, y)$ | Gödelian maximum | $\max(x, y)$ |
| Łukasiewicz intersection | $\max(0, x + y - 1)$ | Łukasiewicz union | $\min(x + y, 1)$ |
| Algebraic product | $x \cdot y$ | Algebraic sum | $x + y - x \cdot y$ |
| Drastic product | $\begin{cases} x \text{ if } y = 1 \\ y \text{ if } x = 1 \\ 0 \text{ otherwise} \end{cases}$ | Drastic sum | $\begin{cases} x \text{ if } y = 0 \\ y \text{ if } x = 0 \\ 1 \text{ otherwise} \end{cases}$ |
| Einstein product | $\frac{x \cdot y}{1+(1-x)\cdot(1-y)}$ | Einstein sum | $\frac{x+y}{1+x \cdot y}$ |

*Fuzzy relation algebra.* A *fuzzy set* $A$ over the universe $\mathcal{U}$ is defined by its characteristic function $\mu_A : \mathcal{U} \to [0, 1]$ mapping all elements of the universe to their respective degree of truth (or membership). This function $\mu_A$ is usually named the *membership function* of $A$. We follow [29] and abbreviate $\mu_A(x)$ to $A(x)$. Wherever, for every $x \in \mathcal{U}$, $A(x) = 0$ or $A(x) = 1$, $A$ is said to be a *crisp set*, representing a classical set. Analogously, a *fuzzy (binary) relation* [38] $R : X \to Y$ between two crisp sets $X$ and $Y$ is characterized by the membership function $\mu_R : X \times Y \to [0, 1]$, specifying the degree of membership of each pair $(x, y)$ in $R$. Again, we abbreviate $\mu_R(x, y)$ by $R(x, y)$. Binary relations can be seen as sets whose elements are pairs, and all operations on sets can be applied to binary relations. Moreover, the concept can be generalized to relations of higher arity.

The standard operations on sets and relations can "go fuzzy" as follows:

- The *complement* of a fuzzy set $A$ is the fuzzy set $\overline{A}$ defined by the membership function $\overline{A}(x) = 1 - A(x)$, for every $x \in \mathcal{U}$.
- Two fuzzy sets $A$ and $B$ may be combined through *intersection* $(A \cap B)(x) = A(x) \sqcap B(x)$ or *union* $(A \cup B)(x) = A(x) \sqcup B(x)$.
- The *Cartesian product* of two fuzzy sets $A$ and $B$ follows the definition of the t-norm, i.e., $(A \times B)(x, y) = A(x) \sqcap B(y)$ for $x, y \in \mathcal{U}$.
- A fuzzy set $A$ is *contained* in another fuzzy set $B$, written $A \subseteq B$, whenever $A(x) \leq B(x)$ for every $x \in \mathcal{U}$.
- Every fuzzy relation $R : X \to Y$ has a *converse* relation $R^\circ : Y \to X$, which is such that $R^\circ(y, x) = R(x, y)$, for all $x, y \in \mathcal{U}$.
- Two fuzzy relations $R : X \to Y$ and $S : Y \to Z$ can be *composed* to create the relation $R \cdot S : X \to Z$ such that $(R \cdot S)(x, z) = (\bigsqcup y : R(x, y) \sqcap S(y, z))$.

A fuzzy operator that arises from the notion of crispness is the *α-cut*. The $\alpha$-cut of a fuzzy set $A$ is the crisp set $A^\alpha = \{a \in \mathcal{U} \mid A(a) \geq \alpha\}$.

## 3 QAlloy-F by Example

$$\begin{tikzcd} Patient \arrow[r, "Q"] \arrow[rr, bend right, "T\supseteq Q\cdot R"'] & Symptom \arrow[r, "R"] & Disease \end{tikzcd}$$

Let us start with a simple, classical fuzzy problem — Sanchez's approach to performing medical diagnosis [29]. Given a fuzzy relation $Q : Patient \to Symptom$ that encodes the symptoms exhibited by a set of patients, and a fuzzy relation $T : Patient \to Disease$ that determines how experts diagnosed those patients, in [29] the authors propose to synthesize the medical knowledge as a fuzzy relation $R : Symptom \to Disease$ such that $Q \cdot R \subseteq T$. This relation could then be applied to other patients, with the disease(s) that display the highest degree selected as diagnoses. This is expected to be an iterative process in which experts validating the candidate $R$ relations could be easily supported by QAlloy-F.

Lines 1–7 of Fig. 1 show how these relations could be encoded in QAlloy-F. The structure is introduced through the declaration of *signatures* and *fields* relating them. Signature `Patient` represents patients. The particular symptoms and diseases present in the collected data are declared as specializations of their parent signatures `Symptom` and `Disease`, respectively, which are marked as **`abstract`** to avoid atoms outside those specializations.

Regular fields in Alloy relate each atom of the parent signature with atoms of other signatures, according to some multiplicity. In QAlloy-F, however, these can be annotated with keyword **`fuzzy`**, denoting that the membership of a tuple in such relations is no longer Boolean but rather a value between 0 and 1. In the example, we have a fuzzy field relating each patient with **`some`** symptoms, and another relating each symptom with a **`set`** of diseases.

The (fuzzy) diagnosis of a patient is simply encoded as `Q.R`, where `.` denotes relational composition, which by default has a Gödelian min-max semantics (recall Table 1): for a patient `p`, a disease `d` and a symptom `s`, select the minimum between the degree `p` exhibits `s` and the degree `s` is related with `d`; then the degree between `p` and `d` is the maximum among all available symptoms.

**Table 2.** Example of a medical diagnosis relation $R$

| $R$ | Viral Fever | Typhoid | Stomach problem | Malaria | Chest problem |
|---|---|---|---|---|---|
| Temperature | 0.4 | 0.3 | 0.1 | 0.7 | 0.1 |
| Cough | 0.4 | 0.2 | 0.2 | 0.7 | 0.2 |
| Stomach pain | 0.1 | 0.2 | 0.8 | 0 | 0.2 |
| Chest pain | 0.1 | 0.1 | 0.2 | 0.1 | 0.8 |
| Headache | 0.3 | 0.6 | 0.2 | 0.2 | 0 |

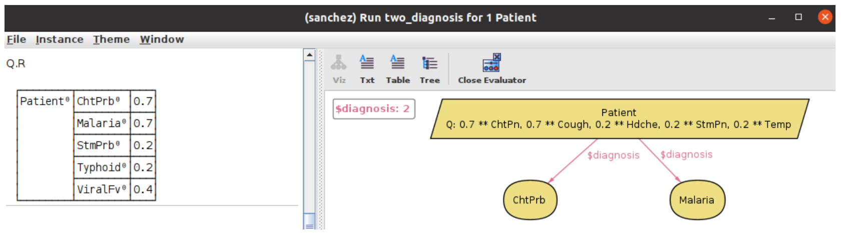
The final diagnosis is obtained by selecting the disease(s) with maximum degree through auxiliary function `diagnosis` (ll. 9–10). It constructs by comprehension a crisp relation between each patient and the diseases diagnosed with maximum degree (function `max` returns a crisp relation with the tuples with maximum degree). Finally, we need to encode the relation $R$ determined by the

```
1  sig Patient {
2    fuzzy Q : some Symptom }
3  abstract sig Symptom {
4    fuzzy R : set Disease }
5  one sig Temp, Cough, StmPn, ChtPn, Hdche extends Symptom {}
6  abstract sig Disease {}
7  one sig ViralFv, Typhoid, StmPrb, Malaria, ChtPrb extends Disease {}
8
9  fun diagnosis : Patient → Disease {
10   { p:Patient, d:Disease | d in max[p.Q.R] } }
11
12 fun expert_R : Symptom → Disease {
13   (0.4**Temp + 0.4**Cough + 0.1**StmPn + 0.1**ChtPn + 0.3**Hdche) → ViralFv +
14   (0.7**Temp + 0.2**Hdche + 0.7**Cough + 0.1**ChtPn) → Malaria +
15   (0.3**Temp + 0.2**Cough + 0.1**ChtPn + 0.6**Hdche + 0.2**StmPn) → Typhoid +
16   (0.1**Temp + 0.2**Cough + 0.8**StmPn + 0.2**ChtPn + 0.2**Hdche) → StmPrb +
17   (0.1**Temp + 0.2**StmPn + 0.8**ChtPn + 0.2**Cough) → ChtPrb }
18
19 run two_diagnosis {
20   R = expert_R and
21   some p:Patient |
22     Malaria + ChtPrb in p.diagnosis } for 1 Patient
23
24 run same_diagnosis {
25   R = expert_R and
26   some p1,p2:Patient |
27     p1.diagnosis = p2.diagnosis and no p1.Q & p2.Q } for 2 Patient
28
29 check maxChestPain {
30   R = expert_R implies all p:Patient |
31     ChtPn in max[p.Q] implies ChtPrb in p.diagnosis }
```

**Fig. 1.** Encoding of the fuzzy medical diagnosis in QAlloy-F

experts. As an example, consider the relation $R$ of Table 2, which is taken from reference [9]. To keep the model flexible and allow for the exploration of alternative solutions, we encode it as a function `expert_R` (ll. 12–17). In QAlloy-F, concrete fuzzy relations are created with scalar multiplication `**`. Here, we create fuzzy sets of symptoms and assign them to diseases through Cartesian product $\rightarrow$. Symptoms with degree 0 are simply absent from the resulting relation.

Given this model, we can make use of QAlloy-F's model finding capabilities to validate the provided $R$. Instances can be generated with **`run`** commands accompanied by further restrictions. For instance, suppose we want to see scenarios where the provided $R$ leads to two diseases being diagnosed for the same patient. The command in ll. 19–22 asks for instances where `R` is exactly the expert data and there is some patient diagnosed with malaria and chest problems.


(sanchez) Run two_diagnosis for 1 Patient

Q.R

|  |  |  |
|---|---|---|
| Patient$^0$ | ChtPrb$^0$ | 0.7 |
|  | Malaria$^0$ | 0.7 |
|  | StmPrb$^0$ | 0.2 |
|  | Typhoid$^0$ | 0.2 |
|  | ViralFv$^0$ | 0.4 |

File Instance Theme Window
Viz
Txt
Table
Tree
Close Evaluator
$diagnosis: 2
Patient
Q: 0.7 ** ChtPn, 0.7 ** Cough, 0.2 ** Hdche, 0.2 ** StmPn, 0.2 ** Temp
$diagnosis
$diagnosis
ChtPrb
Malaria

**Fig. 2.** Instance visualization for model in Fig. 1

`Q` is left unrestricted so that all possible symptom valuations are explored by the finder. For now we focus on scenarios with a single patient, so we restrict the command's scope, which determines how many atoms each signature may contain. Running the command results in the instance shown in Fig. 2, after some theme customization to enhance readability. Interestingly, the patient is mostly showing signs of chest pain and coughing, which lead to 2 diagnosed diseases. The evaluator (left-hand side) can be used to further inspect the instance, whereby it will show that a patient with those symptoms is most likely afflicted by chest problems and malaria, the diseases of maximum degree 0.7. Alternative witnesses to this scenario can also be enumerated from the visualizer.

As another example, consider the command in ll. 24–27, which asks for scenarios where two patients have the same diagnosis but disjoint symptoms (notice the increased scope on patients). The command is valid, and experts would need to validate the scenarios and act accordingly.

Besides encoding scenarios as **`run`** commands, we can also rely on **`check`** commands to check whether bottom-line properties hold. For instance, the command in ll. 29–31 checks whether having chest pain among the most relevant symptoms implies chest problems among the diagnosis. QAlloy-F searches for all possible symptoms for patients and finds no counter-example, meaning that the property is always true for the provided $R$. Experts can encode several of these properties to be automatically checked for alternative $R$ values. Note that we do not have to completely fix `R` with the expert knowledge. Say we want to check whether the property still holds for alternative relationships between chest pain and stomach problems. We could just constrain `R` to a subset of `expert_R` by writing `R − ChtPn→StmPrb = expert_R − ChtPn→StmPrb`, leaving the degree to which chest pain is related to stomach problems free. QAlloy-F will show that the property no longer holds by assigning it a large degree in `R`.

## 4 The QAlloy-F Analyzer

*Language.* The full syntax of the QAlloy-F language is presented in Fig. 3, with changes relative to standard Alloy underlined. Compared to QAlloy-I [30] for the integer domain, the **`int`** keyword was replaced with **`fuzzy`** to declare quantitative relations over the unit interval of real numbers. Much like in the integer domain,

```
spec ::= module qualName [ [ name,+ ] ] import* paragraph*
import ::= open qualName [ [ qualName,+ ] ] [ as name ]
paragraph ::= sigDecl | factDecl | funDecl | predDecl | asrtDecl | chckCmd
sigDecl ::= [ fuzzy ] [ abstract ] [ mult ] sig name,+
                [ sigExt ] { fuzzyDecl,* } [ block ]
sigExt ::= extends qualName | in qualName [ + qualName ]*
mult ::= lone | some | one
decl ::= [ disj ] name,+ : [ disj ] expr
fuzzyDecl ::= [ fuzzy ] decl
factDecl ::= fact [ name ] block
asrtDecl ::= assert [ name ] block
funDecl ::= fun name [ [ decl,* ] ] : expr { expr }
predDecl ::= pred name [ [ decl,* ] ] block
expr ::= const | qualName | @name | this | unOp expr | expr binOp expr
    | expr arrowOp expr | expr [ ! | not ] compareOp expr
    | expr ( => | implies ) expr else expr | quant decl,+ blockOrBar
    | expr [ expr,* ] | ( expr ) | block | { decl,+ blockOrBar }
const ::= none | univ | iden
unOp ::= ! | not | no | mult | set | ~ | * | ^ | #
    | drop | real ** | real−cut
binOp ::= || | or | && | and | <=> | iff | => | implies
    | & | + | − | ++ | <: | :> | .
    | add | sub | mul | div | rem
arrowOp ::= [ mult | set ] → [ mult | set ]
compareOp ::= in | = | != | < | ≤ | > | ≥
letDecl ::= name = expr
block ::= { expr* }
blockOrBar ::= block | | expr
quant ::= all | no | mult
chckCmd ::= check qualName [ scope ]
scope ::= for integer [ but typescope,+ ] | for typescope,+
typescope ::= [ exactly ] integer qualName
qualName ::= [ this/ ] ( name/ )* name
```

**Fig. 3.** Concrete syntax of the QAlloy-F language

numeric values (which are now real numbers belonging to $[0, 1]$) can no longer be stand-alone expressions, but must be associated with the *scalar multiplication* operator (`**`), relying on Alloy's rich type system to perform simple dimensional analysis. **`drop`** can be used to transform fuzzy relations into Boolean ones, taking every non-zero value to 1. A new operator that arises from the fuzzy domain is the $\alpha-$**`cut`** introduced in Section 2, with $\alpha \in [0, 1]$. Although in the integer domain we provided two variants of the join operator that were deemed relevant, there are simply too many variants available for the fuzzy domain. Thus, we have opted to let the user select the desired t-norm (and respective t-conorm), from which the semantics of the join operator is derived.

Similarly to QAlloy-I, the formal semantics of QAlloy-F expressions is determined by a *quantity function* that is defined inductively over relational expressions, as Fig. 4 shows for a fragment of the language. Let $\mathcal{A}$ be the universe of atoms, determined from the scope of the command at hand, and $s$ be a *binding*, i.e., a function that for every free relation $r$ and tuple $t$ with appropriate arity returns its quantity $q = s(r,t)$. The quantity function of a relational expression $\Gamma$ under the binding $s$ is given by $[\![\Gamma]\!]_s$. A tuple $t$ belongs to $\Gamma$ under the binding $s$ iff $[\![\Gamma]\!]_s(t) \neq 0$. Signatures and fields are declared as in QAlloy-I. The semantics of fuzzy relational operations are defined as presented in Section 2, varying according to the t-norm ($\sqcap$) and t-conorm ($\sqcup$) considered. Arithmetic operators are still supported but truncated in the range [0,1], and the scalar multiplication ($**$) now takes a real number $\alpha \in [0,1]$ instead of an integer. The semantics of QAlloy-F formulas is identical to QAlloy-I, for which the reader is redirected to [30]. It is worth noting that QAlloy-F is retro-compatible, so a model with no fuzzy relations/expressions is equivalent to a standard Alloy model.

*Backend.* QAlloy extends Kodkod [34] to support generic *numeric structures* to manage the numeric matrices. This was extended to handle real numbers and represent fuzzy relations, with the matrix operations being further generalized to depend on the definition of t-norms, adding support for those in Table 1. With the quantitative Kodkod problem encoded through fuzzy matrices combined through linear algebra operations, QAlloy-F makes use of SMT solvers to automatically solve the resulting formulas. This SMT problem uses *real function symbols* instead of integer ones, and takes into consideration the definition of the selected t-norm/t-conorm pair. Moreover, the SMT solvers perform verification of these problems according to the *Theory of Reals*, namely over the logic fragment `QF_NRA` (*Quantifier-free real arithmetic*) [4].

*Alloy Analyzer.* The QAlloy-F Analyzer[4] is responsible for parsing the model, encoding it into a quantitative Kodkod problem and then for interpreting back the results obtained and presenting them to the user. It has therefore been adapted in a similar way to QAlloy-I. A key difference is that in this context, the Analyzer provides to the user the various t-norms supported by selecting `T-norm` in the `Options` in the menu bar. Alternatively it can be set through a special annotation in the model, making it self-contained.

After obtaining a solution, the user may further interact with it through the Analyzer features such as the *Visualizer* and *Evaluator*, which were adapted to present the fuzzy information to the user (see Fig. 2) and allow the evaluation of fuzzy expressions, respectively.

## 5 Validating Fuzzy Inference Systems with QAlloy-F

Fuzzy controllers are one of the main applications of fuzzy logic and, in some scenarios, they have shown to be almost as effective as precise mathematical models, being cheaper to develop and implement [28].

[4] QAlloy-F is publicly available at `https://github.com/pf7/QAlloy-F/`.

$$
\begin{array}{ll}
[\![r]\!]_s(t) & = s(r,t) \\
[\![x]\!]_s((a)) & = s(x,(a)) \\
[\![\mathbf{univ}]\!]_s((a)) & = 1 \\
[\![\mathbf{none}]\!]_s((a)) & = 0 \\
[\![\mathbf{iden}]\!]_s((a,b)) & = \begin{cases} 1 \text{ if } a = b \\ 0 \text{ otherwise} \end{cases} \\
[\![\Gamma \texttt{ + } \Delta]\!]_s(t) & = [\![\Gamma]\!]_s(t) \sqcup [\![\Delta]\!]_s(t) \\
[\![\Gamma \texttt{ \& } \Delta]\!]_s(t) & = [\![\Gamma]\!]_s(t) \sqcap [\![\Delta]\!]_s(t) \\
[\![\Gamma - \Delta]\!]_s(t) & = \begin{cases} 0 & \text{if } [\![\Gamma]\!]_s(t) = 0 \\ [\![\Gamma]\!]_s(t) - \min([\![\Gamma]\!]_s(t), [\![\Delta]\!]_s(t)) & \text{otherwise} \end{cases} \\
[\![\mathbf{add}[\Gamma,\Delta]]\!]_s(t) & = min([\![\Gamma]\!]_s(t) + [\![\Delta]\!]_s(t), 1) \\
[\![\mathbf{sub}[\Gamma,\Delta]]\!]_s(t) & = max([\![\Gamma]\!]_s(t) - [\![\Delta]\!]_s(t), 0) \\
[\![\mathbf{mul}[\Gamma,\Delta]]\!]_s(t) & = [\![\Gamma]\!]_s(t) \times [\![\Delta]\!]_s(t) \\
[\![\mathbf{div}[\Gamma,\Delta]]\!]_s(t) & = min([\![\Gamma]\!]_s(t) \;/\; [\![\Delta]\!]_s(t), 1) \\
[\![\mathbf{rem}[\Gamma,\Delta]]\!]_s(t) & = [\![\Gamma]\!]_s(t) \,\%\, [\![\Delta]\!]_s(t) \\
[\![\alpha \texttt{ ** } \Gamma]\!]_s(t) & = \alpha[\![\Gamma]\!]_s(t) \\
[\![\Gamma \texttt{ . } \Delta]\!]_s((a_1,\ldots,a_{n-1},b_2,\ldots,b_m)) & = \bigsqcup_{c \in \mathcal{A}} (([\![\Gamma]\!]_s((a_1,\ldots,a_{n-1},c)) \sqcap [\![\Delta]\!]_s((c,b_2,\ldots,b_m)))) \\
[\![\Gamma \rightarrow \Delta]\!]_s((a_1,\ldots,a_n,b_1,\ldots,b_m)) & = [\![\Gamma]\!]_s((a_1,\ldots,a_n)) \sqcap [\![\Delta]\!]_s((b_1,\ldots,b_m)) \\
[\![\texttt{\~{}}\Gamma]\!]_s((a,b)) & = [\![\Gamma]\!]_s((b,a)) \\
[\![\texttt{\^{}}\Gamma]\!]_s((a,b)) & = \bigsqcup([\![\Gamma]\!]_s((a,b)), [\![\Gamma \texttt{.} \Gamma]\!]_s((a,b)), [\![\Gamma \texttt{.} \Gamma \texttt{.} \Gamma]\!]_s((a,b)), \ldots) \\
[\![\{x_1 \texttt{ : } \Gamma_1, \ldots, x_n \texttt{ : } \Gamma_n \mid \phi\}]\!]_s((a_1,\ldots,a_n)) & = \begin{cases} [\![\Gamma_1]\!]_s((a_1)) \times \ldots \times [\![\Gamma_n]\!]_s((a_n)) & \text{if } s' \models \phi \\ 0 & \text{otherwise} \end{cases} \\
\quad \text{where } s' = s \oplus (x_1,(a_1)) \mapsto [\![\Gamma_1]\!]_s((a_1)) \oplus \ldots \oplus (x_n,(a_n)) \mapsto [\![\Gamma_n]\!]_s((a_n)) & \\
[\![\mathbf{drop}\ \Gamma]\!]_s(t) & = \begin{cases} 0 \text{ if } [\![\Gamma]\!]_s(t) = 0 \\ 1 \text{ otherwise} \end{cases} \\
[\![\alpha{-}\mathbf{cut}\ \Gamma]\!]_s(t) & = \begin{cases} 1 \text{ if } [\![\Gamma]\!]_s(t) \geq \alpha \\ 0 \text{ otherwise} \end{cases} \\
[\![\texttt{\#}\Gamma]\!]_s(t) & = \begin{cases} min(\sum_{t' \in \lceil \Gamma \rceil} [\![\Gamma]\!]_s(t'), 1) & \text{if } t \in \lceil \Gamma \rceil \\ 0 & \text{otherwise} \end{cases}
\end{array}
$$

**Fig. 4.** Semantics QAlloy-F relational expressions ($\mathcal{A}$ is the declared universe, $n$ and $m$ the arity of $\Gamma$ and $\Delta$, respectively)

Such controllers use *Fuzzy Inference Systems* (FIS) for their decision making, a process depicted in Fig. 5. The most popular FIS types are the *Mamdani* [24] and *Takagi-Sugeno-Kang* (Sugeno in short) [32], the former being better suited for humane usage, while the latter is computationally more efficient [28]. This section describes how Mamdani-type FISs can be encoded and validated in QAlloy-F; a Sugeno-type FIS is used in the evaluation in Section 6.

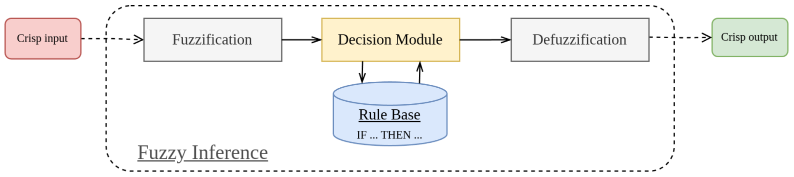


**Fig. 5.** Fuzzy inference process

## 5.1 Developing a FIS for an Automatic Heater

This section describes the development of a fuzzy controller for an automatic heater, using standard tools such as MATLAB®'s Fuzzy Logic Toolbox™.

The controller is expected to react to the temperature and the relative humidity of the environment — input variables $T \in [-20, 50]$ ($°C$) and $H \in [0, 1]$ (%), respectively — and in turn adjust the power level of the heater — output variable $P \in [0, 1]$, ranging from turned off (0) to providing its maximum heat (1). These crisp values can be described through vague terms in the fuzzy realm, known as *linguistic variables*, each represented by a fuzzy set — here, `cold`, `warm` and `hot` for temperature, `dry`, `normal` and `wet` for humidity, and `low`, `mid` and `high` for heater power. This enables reasoning over statements such as "The heater is currently emitting heat at a low power".

At the core of FISs are the so-called *fuzzy rules*. The process starts by *fuzzifying* the crisp values into fuzzy ones through the definition of membership functions. For instance, Fig. 6 shows variables $T$, $H$ and $P$ fuzzified in MATLAB® according to trapezoid, triangular, z- and s-shapes, following the defined linguistic variables. These decisions, which affect the performance of the fuzzy system [1], are commonly made by domain experts or inferred from big data.

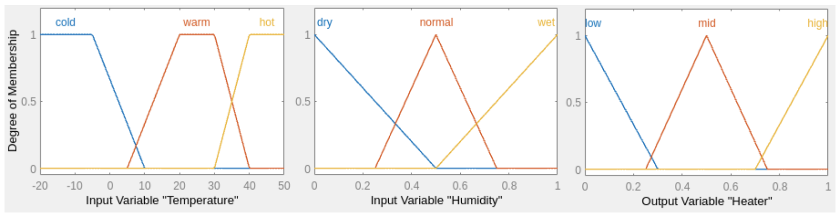
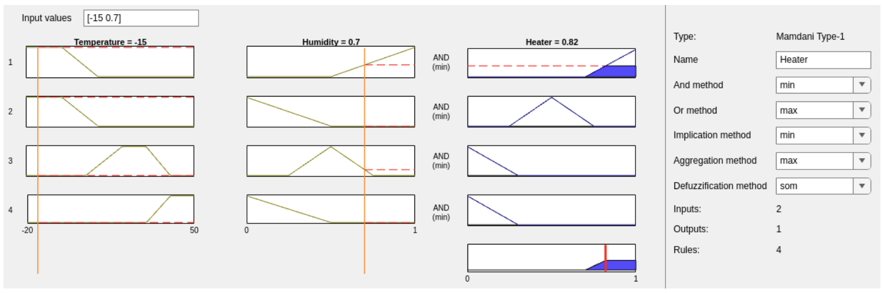


**Fig. 6.** Variables for the automatic heater designed in MATLAB®

The behaviour of the controller is mainly determined by the *rule base* and is also designed with expert insight. An example of a rule is the following, stating that the system must react to cold temperatures and wet air by raising the power to high levels. A rule may depend on multiple antecedents, combined through fuzzy connectives (here, $\sqcap$).

$$\text{IF } T \text{ is } \textit{cold} \text{ AND } H \text{ is } \textit{wet} \text{ THEN } P \text{ is } \textit{high} \tag{R1}$$


Input values [-15 0.7]

Type: Mamdani Type-1
Name: Heater
And method: min
Or method: max
Implication method: min
Aggregation method: max
Defuzzification method: som
Inputs: 2
Outputs: 1
Rules: 4

**Fig. 7.** Rules for the automatic heater designed in MATLAB$^{®}$

The rule inference process is configured to determine the strength of each rule when triggered. This includes selecting operations for calculating the *antecedent* of each rule, an *implication* method to combine the antecedent strength with the output membership function, and an *aggregation* method to combine the output membership function of each rule into a single fuzzy set. Lastly the *defuzzification* step selects a crisp value from the calculated output fuzzy set. This process is shown in Fig. 7 for a system with 4 rules, the first being rule R1.

At this point, the user can select concrete inputs and check the resulting output (-15°$C$ and 70% in Fig. 7). The highly configurable nature of FISs should be clear by now. Note that the parameters of each step affect its behaviour and also its performance [2,15,16]. This motivates the need for tools such as QAlloy-F to complement existing tools for FIS design, by helping the designer validate the fuzzy system and continuously refine it through a back-and-forth between said tools and QAlloy-F, until a satisfactory fuzzy controller is achieved.

## 5.2 Encoding FISs in QAlloy-F

A possible encoding in QAlloy-F of the FIS just described is shown in Fig. 8. The configuration of a FIS developed in MATLAB$^{®}$ is stored in a `.fis` file. We have implemented a prototype, also provided in the QAlloy-F repository, to automatically translate them into such a QAlloy-F model.

QAlloy-F supports Boolean and fuzzy relations only, so it is not possible to directly specify real-numbered crisp values such as e.g. temperatures. We assume that such a range is normalized to the unit interval and $T$ modelled as a fuzzy set, declared with respect to its type `Celsius` (ll. 4–5). Omitted for space-economy, variables $H$ and $P$ are declared analogously as fuzzy sets on `Percentage` and `Power`. The linguistic variables are also defined as signatures, and those for temperature are shown in ll. 7–8. Fuzzy subset signatures `fuzzyT` (l. 9), `fuzzyH` and `fuzzyP` are declared to represent the fuzzified variables. We encoded in a utility module `mf` some function shapes. For instance, by opening `mf` over linguistic variable `Temperature` and crisp value `T` (l. 2), function `trapezoid` returns the corresponding fuzzy input variable according to that shape, as imposed in fact `fuzzification` (ll. 11–16). Note that these shape ranges act on the $[0, 1]$

```
 1 /*# TNORM Godelian #*/
 2 open util/mf[Temperature, T]
 3 ...
 4 one sig Celsius {}
 5 fuzzy sig T in Celsius {}
 6
 7 abstract sig Temperature { R : Humidity set → set Heater }
 8 one sig cold, warm, hot extends Temperature {}
 9 fuzzy sig fuzzyT in Temperature {}
10 ...
11 fact fuzzification {
12   fuzzyT =
13     linz[0.214**cold, 0.429**cold] +
14     trapezoid[0.357**warm, 0.571**warm, 0.714**warm, 0.857**warm] +
15     lins[0.714**hot, 0.857**hot]
16    ...}
17
18 fun ruleBase : Temperature → Humidity → Heater {
19   cold → wet → high +
20   cold → dry → mid +
21   warm → normal → low +
22   hot → dry → low }
23
24 fun aggregated : Heater { fuzzyH.(fuzzyT.R) }
25
26 fact defuzzification {
27   aggregated :> max[aggregated] ≤ fuzzyP :> max[aggregated]
28   no max[aggregated] implies P = 0.5 ** Power }
```

**Fig. 8.** Excerpt of an automatic heater based on Mamdani FIS in QAlloy-F

normalized crisp values rather than on the temperature interval $[-20, 50]$. Our translation from `.fis` configurations automatically performs this normalization.

Rules relate linguistic variables and are naturally represented as a Boolean ternary relation, as declared in l. 7. Function `ruleBase` (ll. 18–22) encodes the provided rule base, with the first entry denoting rule R1. In the decision process we consider only a subset of methods supported in MATLAB® to keep the model succinct and maintainable. In particular, we assume the antecedent and aggregation methods use the selected t-norm (set as a special annotation in l. 1). Thus, by composing the rule base with the fuzzy input variables simultaneously calculates the antecedent with $\sqcap$, and aggregates all rules with $\sqcup$. This is shown in function `aggregated` (l. 24), which returns a fuzzy set on `Heater` linguistic variables. The implication and defuzzification steps are encoded together in a fact given in ll. 26–28. Currently, we only support the maximum defuzzification method by selecting the maximum linguistic variable from `aggregated`. Then, given minimum as the implication method, the fuzzy value of $P$ is forced to take at least said aggregated value. When no rule is triggered, the heater will default to work at mid-power (l. 28), a convention followed by MATLAB®.

### 5.3 FIS Validation and Verification

Given the translation above, one can write commands to simulate the FIS or check for properties. But rather than running the FIS for concrete inputs, we can ask for underspecified scenarios. For instance, the following command asks for instances where the temperature is high but the heater running on low. QAlloy-F quickly answers with an instance with very low level of humidity.

```
run highTemperature {
  ruleBase = R and P < 0.1**Power and T > 0.8**Celsius }
```

We can also test for red flags in our model, for instance: is it ever possible to have the heater run on high power with moderate/hot temperature? QAlloy-F will answer that there is no such instance.

```
check highPower {
  ruleBase = R and T > 0.5**Celsius implies P ≤ 0.5**Power }
```

Perhaps more interestingly, the user can relax the constraints to explore different designs. For instance, expert-information like membership function ranges or the rule base can be partially specified and left for QAlloy-F to explore. Let us say that we want to extend the rule base, and wish to check whether this may break the property above. We can state that `R` is larger than the provided `ruleBase` as follows:

```
check highPowerExtends {
  ruleBase in R and T > 0.5**Celsius implies P ≤ 0.5**Power }
```

QAlloy-F now reports a counter-example: if rule `warm → dry → mid` is added, a low level of humidity will turn the power up.

Given that scenario, we may wish to search for rules where even with low humidity, the power stays low. The following command can be used, which will suggest to use `warm → dry → low` instead.

```
run lowHumidityExtends {
  ruleBase in R and T = 0.5**Celsius and no H and P < 0.5**Power }
```

## 6 Evaluation

We explored the applicability of QAlloy-F to other fuzzy scenarios, leading to the following example models that are also used for performance evaluation:

- `Diagnosis`. The medical diagnosis example presented in Section 3, with the commands presented in Fig. 1.

- `Intuitionistic`. Some authors [9] have argued that the diagnosis problem addressed in Section 3 is better encoded in an *intuitionistic* setting, where each set $A$ has a membership function $\mu_A$ and a *non-membership* function $\nu_A$, related by a score function [7]. QAlloy-F is sufficiently flexible to address this encoding, including intuitionistic composition, which applies regular max-min composition to $\mu_A$ but min-max composition to $\nu_A$. We check commands similar to the non-intuitionistic version, but, perhaps unsurprising, `maxChestPain` is now invalid: even if chest pain has maximum membership degree, when the non-membership degree outweighs it, it is unlikely that the patient is affected by chest problems.
- `Portrait`. Boolean equivalence relations can be used to partition the universe at hand. Similarly, a fuzzy equivalence relation can be transformed into a Boolean one by performing $\alpha-$**`cut`**s over it, with different $\alpha$ values yielding different clusters. This approach has been applied to determine groups of portraits based on visual similarities [33]. We model fuzzy equivalence relations in QAlloy-F, and use these to determine clusters according to desirable characteristics (**`run`** `threeToTwo`); as well to check that increasing the $\alpha-$**`cut`** results in finer-grained clusters (**`check`** `invProportionality`).
- `Mamdani`. The automatic heater Mamdani FIS model described in Section 5, with an $\sqcap$-based rule base, with the commands presented in Section 5.3.
- `Sugeno`. A classical FIS example[5] for determining the tip in regards to the service quality and food taste. We model it as a Sugeno FIS, and in contrast to `Mamdani`, we take a $\sqcup$-based rule base, which is not only modelled through a Boolean relation `R`, but also with a fuzzy relation `W` for the antecedent step and a fuzzy relation `Y` to model its linear/constant rule output functions, both of which are used to model the defuzzification weighted average method in the end. We use QAlloy-F to evaluate different scenarios (e.g. **`run`** `findGenerousTip`) or to evaluate the strength of known rules (e.g. **`check`** `cheapTip`) for example.

Our evaluation of QAlloy-F then aimed to answer the following questions:

**RQ1** Is the analysis of QAlloy-F models feasible?
**RQ2** What is the impact of choosing different t-norms?
**RQ3** What is the impact of choosing different SMT solvers?

To answer these questions we measured the execution time of various commands for the examples described above, for every SMT solver currently integrated in QAlloy-F — Z3 [25] (v4.8.18), MathSAT [8] (v5.6.6), CVC4 [5] (v1.8) and Yices [11] (v2.6.4) — and for the algebraic product, Gödelian, and Łukasiewicz t-norms. All tests were run in a commodity octa-core 3.2GHz AMD Ryzen™ 7 5800H with 16GB of RAM; and QAlloy-F running with 8192MB maximum memory and 8192k of maximum stack size, with a timeout of 10mins. The models, the benchmark script and the full results are available in the QAlloy-F repository. Table 3 presents an excerpt of the results.

[5] `https://www.mathworks.com/help/fuzzy/fuzzy-inference-process.html`

Concerning **RQ1**, our analysis procedures seem to be feasible even for more complex models such as `Mamdani` and `Sugeno`, with all commands solved in a few seconds for some solver and t-norm. Interestingly, there is no clear evidence that satisfiable problems perform better than unsatisfiable ones. Regarding **RQ2**, it is interesting to see that the outcome of some commands actually changes depending on the selected t-norm. In particular, the `maxChestPain` discussed in Section 3 is only valid under the default Gödelian. In terms of performance there is no t-norm clearly outperforming the others, and it often depends on the selected SMT solver. Nonetheless, the algebraic product seems to be the t-norm that most often leads to timeouts in all solvers. As for **RQ3**, there is also no clear SMT solver outperforming the others, but Z3 and MathSAT seem to be overall the most consistent when it comes to arriving at a result in usable times.

**Table 3.** Evaluation results (in ms), entry omitted if all TO, best times in bold, entries associated with "unknown" responses are crossed out

| Model | Cmd | Scope | T-norm | Result | Z3 | MSAT | CVC4 | Yices |
|---|---|---|---|---|---|---|---|---|
| `Diagnosis` | `same_diagnosis` | 2 | Gödelian | SAT | 5703 | **641** | 12162 | TO |
| | | | **Łukasiewicz** | SAT | 627 | **160** | 2697 | TO |
| | `maxChestPain` | 4 | Gödelian | UNSAT | 19765 | **1097** | 20170 | TO |
| | | | Łukasiewicz | SAT | 4463 | **164** | 3802 | 313710 |
| | | | **Product** | SAT | **109** | TO | TO | 264 |
| `Intuitionistic` | `same_diagnosis` | 2 | Gödelian | SAT | 16402 | **11255** | 20569 | 293413 |
| | | | **Łukasiewicz** | SAT | 17276 | **198** | 2448 | 258994 |
| | `maxChestPain` | 1 | Gödelian | SAT | 479 | **423** | 1503 | 14455 |
| | | | **Łukasiewicz** | SAT | 15306 | **97** | 611 | 8742 |
| `Portrait` | `threeToTwo` | 4 | **Gödelian** | SAT | 718 | 485 | 3905 | **314** |
| | | | Łukasiewicz | SAT | 679 | 488 | 3923 | **325** |
| | | | Product | SAT | 672 | 490 | 3930 | **316** |
| | `invProportionality` | 6 | Gödelian | UNSAT | 1313 | **182** | 61969 | 201 |
| | | | Łukasiewicz | UNSAT | 2321 | **150** | 2047 | 701 |
| | | | **Product** | UNSAT | 302 | 158 | 5302 | **143** |
| `Mamdani` | `highPower` | NA | Gödelian | UNSAT | 263 | **104** | 353 | 121 |
| | | | Łukasiewicz | UNSAT | 603 | **61** | 173 | 2894 |
| | | | **Product** | UNSAT | **41** | TO | 228 | 194 |
| | `highPowerExtends` | NA | Gödelian | SAT | 272 | **137** | 726 | 2877 |
| | | | **Łukasiewicz** | SAT | 5375 | **43** | 280 | 2042 |
| | | | Product | SAT | TO | TO | **243** | 1189 |
| `Sugeno` | `findGenerousTip` | 3 | Gödelian | SAT | 2771 | **2202** | TO | 55676 |
| | | | **Łukasiewicz** | SAT | **430** | 453 | TO | TO |
| | | | Product | UNK | ~~54214~~ | ~~290755~~ | TO | TO |
| | `cheapTip` | 4 | Gödelian | UNSAT | **7025** | 7551 | TO | 25303 |
| | | | **Łukasiewicz** | UNSAT | **1863** | 2009 | TO | TO |
| | | | Product | UNSAT | **52852** | ~~342107~~ | TO | TO |

## 7 Related Work

Previous work has relied on SMT solvers to verify formulas in fuzzy logics. Reference [3] describes how SMT solvers can be used to automatically prove formulas of infinitely-valued logics. Reference [37] generalizes this approach to more logics, in particular, continuous t-norm-based logics. Reference [36] further extends [37] to support modal fuzzy logic. The QAlloy backend interprets the fuzzy relational operators as a linear algebra over fuzzy matrices and converts them into SMT formulæ according to the selected t-norm, similarly to these approaches. (They are also considered within fragments of the theory of reals.) But unlike QAlloy, these techniques do not provide a specification language to encode fuzzy models. FLOPER [6] instead relies on fuzzy logic programming to derive all valid models of a formula. It is implemented in Prolog and supports truth degrees defined as a complete bounded lattice $\mathcal{L}$. The authors of [40] describe ongoing work to verify fuzzy logic models, introducing an approach based on symbolic execution, whose prototype also makes use of an SMT solver.

Considerable work has been done on model checking fuzzy systems, including checking branching-time temporal logics over fuzzy Kripke structures [26,31,12,23], branching-time [21,20] and linear-time [19,18,22] temporal logics over possibilistic Kripke structures (which extend fuzzy logic by considering both a possibility and necessity degree). In [35,10] a modified version of the CMur$\varphi$ model checker is proposed for the verification of fuzzy control systems, which relies on external C/C++ functions from the controller. QAlloy is currently focused on the validation of the structural part of fuzzy systems, but extending QAlloy to the temporal capabilities of Alloy 6 is planned future work, as described below.

## 8 Conclusions and Future Work

By the very nature of the underlying logic, fuzzy systems are challenging to design and validate. This paper proposes QAlloy-F, a specification language for fuzzy relational models, backed by automatic analysis procedures for validation and verification in the tradition of Alloy. We show that the language is sufficiently rich to encode fuzzy inference systems, and our evaluation shows the analysis procedures to be performant for models of this complexity.

There are still open issues with quantitative relational model finding that we plan to address. We expect to unify QAlloy-F and QAlloy-I to allow reasoning about integer-valued models with uncertainty. We also intend to integrate the temporal capabilities of Alloy 6 into QAlloy. The techniques implemented in standard Alloy for solution iteration are not effective in generating varied quantitative solutions, so symmetry breaking must be addressed in this context.

**Acknowledgements**

The work by J.N. Oliveira is financed by National Funds through the FCT - Fundação para a Ciência e a Tecnologia, I.P. (Portuguese Foundation for Science and Technology) within the project IBEX, with reference PTDC/CCI-COM/4280/2021. The work by P. Silva, subject to the PhD studentship grant with reference 2023.01186.BD, is financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project LA/P/0063/2020. DOI 10.54499/LA/P/0063/2020.

## References

1. Adil, O., Ali, A., Sumait, B.: Comparison between the effects of different types of membership functions on fuzzy logic controller performance. International Journal of Emerging Engineering Research and Technology **3**, 76–83 (04 2015)
2. Ahmad, K., Mesiarova, A.: Choosing t-norms and t-conorms for fuzzy controllers. vol. 2, pp. 641 – 646 (09 2007). https://doi.org/10.1109/FSKD.2007.216
3. Ansótegui, C., Bofill, M., Manyà, F., Villaret, M.: Building automated theorem provers for infinitely-valued logics with satisfiability modulo theory solvers. In: 2012 IEEE 42nd International Symposium on Multiple-Valued Logic. pp. 25–30 (2012). https://doi.org/10.1109/ISMVL.2012.63
4. Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org` (2016)
5. Barrett, C.W., Conway, C.L., Deters, M., Hadarean, L., Jovanovic, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6806, pp. 171–177. Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_14, `https://doi.org/10.1007/978-3-642-22110-1_14`
6. Bofill, M., Moreno, G., Vázquez Pérez-Íñigo, C., Villaret, M.: Automatic proving of fuzzy formulae with fuzzy logic programming and SMT. Proceedings of XIII Spanish Conference on Programming and Languages, PROLE 2013 (01 2014). https://doi.org/10.14279/tuj.eceasst.64.991.974
7. Chen, T.: A comparative analysis of score functions for multiple criteria decision making in intuitionistic fuzzy settings. Inf. Sci. **181**(17), 3652–3676 (2011). https://doi.org/10.1016/J.INS.2011.04.030, `https://doi.org/10.1016/j.ins.2011.04.030`
8. Cimatti, A., Griggio, A., Schaafsma, B., Sebastiani, R.: The MathSAT5 SMT Solver. In: Piterman, N., Smolka, S. (eds.) Proceedings of TACAS. LNCS, vol. 7795. Springer (2013)
9. De, S.K., Biswas, R., Roy, A.R.: An application of intuitionistic fuzzy sets in medical diagnosis. Fuzzy Sets Syst. **117**(2), 209–213 (2001)
10. Della Penna, G., Intrigila, B., Magazzeni, D.: Evaluating fuzzy controller robustness using model checking. In: Di Gesù, V., Pal, S.K., Petrosino, A. (eds.) Fuzzy Logic and Applications. pp. 303–311. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
11. Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) Computer Aided Verification. pp. 737–744. Springer International Publishing, Cham (2014)
12. Ebrahimi, M., Sotudeh, G., Movaghar, A.: Symbolic checking of fuzzy ctl on fuzzy program graph. Acta Informatica **56** (02 2019). https://doi.org/10.1007/s00236-018-0311-3
13. Jackson, D.: Alloy: A language and tool for exploring software designs. Communications of the ACM **62**(9), 66–76 (2019). https://doi.org/https://doi.org/10.1145/3338843
14. Jang, J., Gulley.: Matlab: Fuzzy logic toolbox user's guide. the math-works, inc., natick, 19-127. (1997)
15. Kiszka, J.B., Kochańska, M.E., Sliwińska, D.S.: The influence of some fuzzy implication operators on the accuracy of a fuzzy model-part i. Fuzzy Sets and Systems **15**(2), 111–128 (1985). https://doi.org/https://doi.org/10.1016/0165-0114(85)90041-7, `https://www.sciencedirect.com/science/article/pii/0165011485900417`

16. Kiszka, J.B., Kochańska, M.E., Sliwińska, D.S.: The influence of some fuzzy implication operators on the accuracy of a fuzzy model-part ii. Fuzzy Sets and Systems **15**(3), 223–240 (1985). https://doi.org/https://doi.org/10.1016/0165-0114(85)90016-8, https://www.sciencedirect.com/science/article/pii/0165011485900168
17. Kumar, S., Gangwal, C.: A study of fuzzy relation and its application in medical diagnosis. Asian Research Journal of Mathematics pp. 6–11 (06 2021). https://doi.org/10.9734/arjom/2021/v17i430289
18. Li, Y.: Quantitative model checking of linear-time properties based on generalized possibility measures. Fuzzy Sets Syst. **320**, 17–39 (2017)
19. Li, Y., Li, L.: Model checking of linear-time properties based on possibility measure. IEEE Trans. Fuzzy Syst. **21**(5), 842–854 (2013)
20. Li, Y., Li, Y., Ma, Z.: Computation tree logic model checking based on possibility measures. Fuzzy Sets Syst. **262**, 44–59 (2015)
21. Li, Y., Ma, Z.: Quantitative computation tree logic model checking based on generalized possibility measures. IEEE Trans. Fuzzy Syst. **23**(6), 2034–2047 (2015)
22. Li, Y., Wei, J.: Possibilistic fuzzy linear temporal logic and its model checking. IEEE Transactions on Fuzzy Systems **29**(7), 1899–1913 (2021). https://doi.org/10.1109/TFUZZ.2020.2988848
23. Ma, Z., Li, Z., Li, W., Gao, Y., Li, X.: Model checking fuzzy computation tree logic based on fuzzy decision processes with cost. Entropy **24**(9) (2022). https://doi.org/10.3390/e24091183, https://www.mdpi.com/1099-4300/24/9/1183
24. Mamdani, E., Assilian, S.: An experiment in linguistic synthesis with a fuzzy logic controller. International Journal of Man-Machine Studies **7**(1), 1–13 (1975). https://doi.org/https://doi.org/10.1016/S0020-7373(75)80002-2, https://www.sciencedirect.com/science/article/pii/S0020737375800022
25. de Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
26. Pan, H., Li, Y., Cao, Y., Ma, Z.: Model checking fuzzy computation tree logic. Fuzzy Sets and Systems **262**, 60–77 (2015). https://doi.org/https://doi.org/10.1016/j.fss.2014.07.008, https://www.sciencedirect.com/science/article/pii/S0165011414003157, theme: Logic and Computer Science
27. Rada-Vilela, J.: The fuzzylite libraries for fuzzy logic control (2018), https://fuzzylite.com/
28. Reznik, L.: Fuzzy controllers handbook: how to design them, how they work. Elsevier (1997)
29. Sanchez, E.: Solutions in composite fuzzy relation equations: Application to medical diagnosis in brouwerian logic. In: Dubois, D., Prade, H., Yager, R.R. (eds.) Readings in Fuzzy Sets for Intelligent Systems, pp. 159–165. Morgan Kaufmann (1993). https://doi.org/https://doi.org/10.1016/B978-1-4832-1450-4.50017-1, https://www.sciencedirect.com/science/article/pii/B9781483214504500171
30. Silva, P., Oliveira, J.N., Macedo, N., Cunha, A.: Quantitative relational modelling with QAlloy. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. p. 885–896. ESEC/FSE 2022, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3540250.3549154, https://doi.org/10.1145/3540250.3549154

31. Sotudeh, G., Movaghar, A.: Abstraction and approximation in fuzzy temporal logics and models. Formal Aspects Comput. **27**(2), 309–334 (2015)
32. Takagi, T., Sugeno, M.: Fuzzy identification of systems and its applications to modeling and control. IEEE Transactions on Systems, Man, and Cybernetics **SMC-15**(1), 116–132 (1985). https://doi.org/10.1109/TSMC.1985.6313399
33. Tamura, S., Higuchi, S., Tanaka, K.: Pattern classification based on fuzzy relations. IEEE Transactions on Systems, Man, and Cybernetics **SMC-1**(1), 61–66 (1971). https://doi.org/10.1109/TSMC.1971.5408605
34. Torlak, E., Jackson, D.: Kodkod: A relational model finder. In: Grumberg, O., Huth, M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 632–647. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
35. Tronci, E., Melatti, I., Tofani, A., Magazzeni, D., Intrigila, B., Tronci, E., Melatti, I., Tofani, A., Magazzeni, D., Intrigila, B.: A model checking technique for the verification of fuzzy control systems. In: International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06). vol. 1, pp. 536–542 (2005). https://doi.org/10.1109/CIMCA.2005.1631319
36. Vidal, A.: MNiBLoS: A SMT-based solver for continuous t-norm based logics and some of their modal expansions. Information Sciences **372**, 709–730 (2016). https://doi.org/https://doi.org/10.1016/j.ins.2016.08.072, `https://www.sciencedirect.com/science/article/pii/S0020025516306491`
37. Vidal, A., Bou, F., Godo, L.: An SMT-based solver for continuous t-norm based logics. In: Hüllermeier, E., Link, S., Fober, T., Seeger, B. (eds.) Scalable Uncertainty Management. pp. 633–640. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
38. Winter, M.: Goguen Categories—A Categorical Approach to L-Fuzzy Relations. No. 25 in Trends in Logic, Springer-Verlag (2007)
39. Zadeh, L.: Fuzzy sets. Information and Control **8**(3), 338–353 (1965). https://doi.org/https://doi.org/10.1016/S0019-9958(65)90241-X, `https://www.sciencedirect.com/science/article/pii/S001999586590241X`
40. Zhao, S., Li, Z., Chen, Z., Wang, J.: Symbolic verification of fuzzy logic models. pp. 1787–1789 (09 2023). https://doi.org/10.1109/ASE56229.2023.00087