Acta Informatica (2019) 56:1–33
https://doi.org/10.1007/s00236-018-0311-3

ORIGINAL ARTICLE

Symbolic checking of Fuzzy CTL on Fuzzy Program Graph

Masoud Ebrahimi[1] · **Gholamreza Sotudeh**[2] · **Ali Movaghar**[3]

Received: 16 May 2016 / Accepted: 9 January 2018 / Published online: 3 February 2018

Abstract Few fuzzy temporal logics and modeling formalisms are developed such that their model checking is both effective and efficient. State-space explosion makes model checking of fuzzy temporal logics inefficient. That is because either the modeling formalism itself is not compact, or the verification approach requires an exponentially larger yet intermediate representation of the modeling formalism. To exemplify, Fuzzy Program Graph (FzPG) is a very compact, and powerful formalism to model fuzzy systems; yet, it is required to be translated into an equal Fuzzy Kripke model with an exponential blow-up should it be formally verified. In this paper, we introduce Fuzzy Computation Tree Logic (FzCTL) and its direct symbolic model checking over FzPG that avoids the aforementioned state-space explosion. Considering compactness and readability of FzPG along with expressiveness of FzCTL, we believe the proposed method is applicable in real-world scenarios. Finally, we study formal verification of fuzzy flip-flops to demonstrate capabilities of the proposed method.

The majority of this work was done before the M. Ebrahimi joined Graz University of Technology. At Graz University of Technology, M. Ebrahimi is funded by the LEAD project "Dependable Internet of Things in Adverse Environments".

✉ Masoud Ebrahimi
masoud.ebrahimi@iaik.tugraz.at

Gholamreza Sotudeh
setoodeh@iaushiraz.ac.ir

Ali Movaghar
movaghar@sharif.edu

[1] Graz University of Technology, Graz, Austria

[2] Department of Computer Engineering, Azad University of Shiraz, Shiraz, Iran

[3] Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

1 Introduction

The real world phenomena will result in human assertions that are imprecise and uncertain; for instance, measuring quantities in the real world can lead to imprecise, fuzzy, or even vague assertions. One cannot simply cope with these types of fuzziness, vagueness, and uncertainties using classical logics; therefore, new types of theories and logics are invented to tackle the issues arising from uncertainties and imprecisions. Among these newly invented theories and logics, probabilistic theory and fuzzy logic together can cope with most of those issues. However, in order to understand what types of uncertainties can be dealt with probability theory and what aspects of uncertainties are best handled with fuzzy logic, it is best for one to know about the background of probability theory and fuzzy logic and their application domains.

Probability theory is defined to deal with uncertainties arising from random events while fuzzy logic is the theory to deal with the data demonstrating uncertainties other than randomness, like fuzziness and sometimes vagueness in expression. The late professor Lotfi Zadeh discussed some fundamental shortcomings in classical probability theory and took several issues with the inclusion of fuzzy logics by this theory. Finally, he specifically stated that the two concepts are complementary; yet, a perception-based probability theory is more general and more complex than probability theory, conceptually, mathematically, and computationally [29]. Consequently, throughout this paper we neither intend to elaborate on fuzzy models nor plan to advocate fuzzy logic; instead, we assume the importance of having fuzzy systems in parallel with probabilistic systems is already justified.

Model based design and formal verification are getting an ever-increasing attention in system development. It is common for a system design, should it be formally verified, to be represented by an appropriate mathematical model, while its behavior is expressed by a descriptive specification framework; thereupon, one is able to devise an algorithm to systematically analyze if the model and subsequently the system's behavior guarantees certain properties. Since various stochastic phenomena are involved in reliability of a system, verifying if a system design absolutely guarantees certain properties is a rigid notion. Subsequently, focusing on absolute guaranties while verifying system's behavior is not empirical.

System's behavior can be considered uncertain if its reliability is susceptible to external stimuli that cannot be controlled or prevented. In order to have an empirical verification approach for systems with uncertain behavior, the mathematical model of the system must be adapted to grasp certain degrees of uncertainty. Meanwhile, the specification framework shall also possess correct understanding of uncertainty at the same level. Consequently, verification approaches are invented to verify certain aspects of systems that show stochastic, random, and uncertain behaviors.

Systems that are subject to or dealing with random phenomena are comprising both nondeterministic and probabilistic state transitions simultaneously. Markovian models typically represent these systems. A discrete-time finite state transition system whose transition to next state is distributed probabilistically is a discrete-time Markov Chain. A more sophisticated Markovian model in which nondeterministic and probabilistic transitions coexist is called a Markov Decision Process. In in any state of Markov decision process there exist a nondeterministic choice between probability distributions for the successors.

Probabilistic Computation Tree Logic (PCTL) is an extension of CTL such that it formulates conditions on a state of a Markov chain and Markov decision process [2,4,10]. In PCTL, a quantitative counterpart replaced path quantifiers $\exists$ and $\forall$ that is denoted as $\mathbb{P}_J(\varphi)$, where J indicates a probabilistic bound on the number of paths satisfying property φ and

yet the interpretation of a PCTL formula remains Boolean; that is, a state either satisfies or violates a PCTL formula. PCTL model checking for Markov chains is a natural extension of CTL model checking. A state s is considered in satisfaction set for a state formula only if the probability of s satisfying certain path property is in the probabilistic bound J. Unfortunately Markov chains and Markov decision processes are not powerful enough to model sophisticated dynamics of systems.

A more sophisticated, and comprehensive class of systems are hybrid systems that consist both discrete and continuous components. In order to model hybrid systems, Henzinger introduced Hybrid Automata [11]. Hybrid automata grasp the nature of systems that combine discrete and continuous dynamics by combining continuously evolving variables and finite state control graphs. The discrete components in hybrid automata are usually digital components that regulate and interact with analog components. Analog sampling and timing are important in hybrid systems and subsequently in hybrid automata for state transitions can occur discretely and instantaneously in the control graph or continuously as time passes by.

Although hybrid automata is very powerful in modeling hybrid systems, its verification problem is generally undecidable even for simple safety properties [12]. Therefore, studies on hybrid systems focus on classes of hybrid automata where the safety verification problem is decidable. Timed Automata are a subclass of hybrid automata whose reachability is decidable. Timed automata consist continuous variables called clocks that continuously evolve with constant slope 1. Clocks are frequently compared to constants and sometimes reset to 0. An extension of timed automata incorporates probabilistic theory to model systems that deal with random phenomena [18].

Probabilistic timed automata is also augmented with additional continuous variables in the form of costs or rewards; the resulting automata is often referred to as priced probabilistic timed automata. In this class of automata, all state and transitions are labeled with state and action rewards, respectively. PCTL is augmented with operators to reason about rewards by comprising a reward operator whose interpretation is Boolean; that is, whenever the expected value of a reward function on a reward/cost variable belongs to certain continuous bound it evaluates to 1 and otherwise to 0. A verification approach is also devised for this class of timed automata and proved to be useful in practice [18]; yet, it is only suitable in formal verification of probabilistic systems.

While modeling, specifying, and verifying probabilistic systems is well studied, verification problem for another subclass of hybrid/dynamic systems that incorporates fuzzy logic is overlooked. Hybrid automata can also model fuzzy logic controllers, as they are a class of hybrid systems. Fuzzy logic control systems can be cast in terms that human operators can understand and better relate to; thus, user's experience can be used in design phase of the controller [23]. Consequently, fuzzy logic control systems are very popular because they mechanized most of tasks that humans can successfully perform. Meanwhile, numerous fuzzy logic models are developed to represent the underlying logic in fuzzy control systems; yet, few efforts are made to bridge the gap between system modeling and behavior specification, and from thereon to verification.

Seemingly, a scenario quite similar to the narrative of probabilistic system verification is also applicable in formal verification of fuzzy logic systems. It begins with formal verification of discrete-time fuzzy logic models of corresponding systems; then, higher-level abstractions and augmented models shall be built on top of discrete-time fuzzy logic models to address more sophisticated classes of systems. Finally yet importantly, formal verification of fuzzy logic systems shall encompass continuous time through models like fuzzy logic extended timed automata. Meanwhile, the specification framework shall also go through the same development cycle so that it corresponds to the modeling approach.

To this date, there are few efforts to define a suitable fuzzy temporal logic; among which, some used formal statement as the specification framework [9,17] while others used linguistic statements [3,27]. Some variety of fuzzy extended temporal logics considered the concept of time to be discrete [5,21] while others presents events in continuous time [3,17,27]. In most cases, fuzzy extended temporal logics accompany a trace checking method instead of a formal verification approach. Fuzzy Branching Temporal Logic (FBTL) by Moon et al. [17] is by far the most notable effort towards a specification framework for fuzzy logic systems. Unfortunately, no one could devised an applicable model checking method for FBTL yet. If such an approach should happen to contrive then it will be based on a simpler fuzzy extended temporal logic defined in discrete time [26]. Therefore, in this paper, we are providing more comprehensive semantics in formal verification of discrete-time fuzzy systems.

Fuzzy Markov Chain is one of the notable discrete-time fuzzy logic models that has a finite convergence to a stationary solution [1]. It is also more robust with respect to small perturbations of the transition matrix compared to probabilistic Markov chains. Although fuzzy Markov chain is a good candidate to model discrete-time fuzzy logic systems, we simply believe this is a matter of preference. This paper neither aims nor intends to compare discrete-time fuzzy logic models but since semantics of our specification language is defined based on this structure [26] and we simply do not intend to reinvent the wheel, FzKripke structure is our choice of model for discrete-time fuzzy systems. FzKripke structure is capable of modeling fuzzy logic controllers, and fuzzy logic circuitry. It was even used for modeling fuzzy flip-flops, and a medical diagnosis and treatment example [22,26].

Sotudeh also defined a higher-level abstraction of FzKripke structure that is more compact, and easier to manually manipulate called Fuzzy Program Graph (FzPG) [26]. Although FzPG is more efficient in modeling phase, its verification requires a direct translation to an equal FzKripke structure resulting in an exponential blow-up. Due to quantization of truth degrees, state-space explosion in a fuzzy extended model checking is more plausible compared to the classical model checking. Main contribution of this paper is a symbolic checking method for Fuzzy CTL (FzCTL) formulae over FzPG that avoids translation to an equal FzKripke structure. This saves an exponential blow-up, which is most effective in efficient model checking of complex fuzzy logic systems.

The rest of this paper is as follows: in Sects. 2 and 3, we study quantized FzKripke structure and FzCTL, respectively. Quantized FzPG and symbolic checking of FzCTL over FzPG are studied in Sects. 4 and 5, respectively. We analyze computational and memory complexity of proposed symbolic method; yet, an alternative symbolic checking method for FzCTL over FzPG is studied in Appendix A that is of lower memory complexity. Section 6 provides a case study on applicability of the proposed method in design and verification of fuzzy flip-flops. Finally, in Sect. 7 we conclude and plan our future research.

2 Quantized Fuzzy Kripke model

Fuzzy Kripke Model (FzKripke) Structure [26] is basically χKripke Model [6] whose quasi-boolean algebra is defined on infinite lattice of interval [0, 1]. Similarly the Quantized FzKripke (Q-FzKripke) can be defined over quantized [0, 1]. This model is quite similar to Fuzzy Graph previously introduced in [28]. There are multiple definitions of Fuzzy extended Kripke structure in the literature [22,26], all of which are equal to some extent, but few of which are accompanied with a well-defined fuzzy extended temporal logic so that they can be used in modeling fuzzy control project and fuzzy logic circuitry.

Definition 1 Throughout this paper, $[0, 1]_\Delta$ denotes the quantization of interval $[0, 1]$ and is defined as follows:

$$[0, 1]_\Delta = \left\{ k\Delta : k \in \{0, \ldots, \frac{1}{\Delta}\} \right\}, \text{ where } \Delta \in \frac{1}{\mathbb{N}} .$$

Definition 2 (*Quantized Fuzzy Kripke*) This is a tuple $M = (S, X, \mathrm{R}, \mathrm{L}, \mathrm{I})$ where $S = \{s_1, \ldots, s_n\}$ is a set of states, $X = \langle x_1, \ldots, x_m \rangle$ is a set of attributes, each of which are assigned with a truth degree in $[0, 1]_\Delta$. Function $\mathrm{Val}_\Delta(X)$ expresses different possible values for all attributes as a whole as follows:

$$\mathrm{Val}_\Delta(X) = \left\{ \langle v_1, \ldots, v_m \rangle | \; v_i \in [0, 1]_\Delta \right\}$$

Similarly, for attribute evaluation, a dot operator provides access to the value of each attribute as follows:

$$\mu \in \mathrm{Val}_\Delta(X), \; \mu = \langle v_1, \ldots, v_m \rangle \Rightarrow v_i = \mu \cdot x_i$$

Relation "R" defines the possibility of transition from one state to another, function "L" assigns a label to each state as the state's specific valuation, and function "I" assigns the entrance possibility for each state in the initial step.

$$\begin{aligned} \mathrm{R} &: S \to \mathrm{Val}_\Delta(X) \\ \mathrm{L} &: S \times S \to [0, 1]_\Delta \\ \mathrm{I} &: S \to [0, 1]_\Delta \end{aligned}$$

Definition 3 Given $M = (S, X, \mathrm{R}, \mathrm{L}, \mathrm{I})$, finite and infinite execution paths starting from state s_i are defined as follows:

$$\pi \in \mathrm{Path}_{\mathrm{fin}}(s_i) \Leftrightarrow \pi \in s_i \xrightarrow{r_i} s_{i+1} \xrightarrow{r_{i+1}} \ldots \xrightarrow{r_{i+u-1}} s_{i+u}$$

$$\pi \in \mathrm{Path}_{\mathrm{inf}}(s_i) \Leftrightarrow \pi \in s_i \xrightarrow{r_i} s_{i+1} \xrightarrow{r_{i+1}} s_{i+2} \xrightarrow{r_{i+2}} \ldots$$

$$\forall i, \alpha \in \mathbb{N} \cdot r_{i+\alpha} \in [0, 1]_\Delta$$

Definition 4 Single-Source Quantized Fuzzy Kripke is a Q-FzKripke such that $\mathrm{I}(s_0) = 1$ and $\forall s \in S \setminus \{s_0\} \cdot \mathrm{I}(s) = 0$.

An arbitrary Q-FzKripke can be casted to a single-source one using the method presented for FzKripke in [26]. In order to model check proposition φ on a multi-source Q-FzKripke M, it is enough to check the proposition $\mathrm{AX}(\varphi)$ on the equivalent single-source Q-FzKripke M'.

The reason behind casting a multi-source FzKripke structure to a single-source one is that bisimulation equivalence among FzKripke structures is defined on single-source FzKripke structures [26]. As a reminder it is proved that any CTL* formula evaluates to the same value on bisimilar Kripke structures [2]. Sotudeh and Movaghar also showed that given two bisimilar FzKripke structures an FzCTL* formula is evaluated to a similar truth degree on both of them [26]. Later, we are going to use this property of bisimilar FzKripke structures in devising our symbolic model checking approach.

3 Fuzzy Computational Tree Logic

Throughout this section, we assume the reader has knowledgeable about temporal logics, specifically the Computational Tree Logic (CTL). For a thorough introduction on CTL, we suggest to see relevant chapters in [2].

A reactive system is described adequately only if its ongoing behavior in any time step is addressed. An infinite sequence of actions leaving from current, and leading to next states represents a behavior of the system. Temporal logic is a well-established concept capable of describing behaviors of reactive systems [24]. Given analogue inputs as continuous variables whose range is [0, 1], a fuzzy control system is a reactive system that controls complex and continuously varying systems. Consequently, if one aims to describe and study behaviors of fuzzy control systems a fuzzy extended temporal logic can ease the process.

Since in a fuzzy system we are uncertain about the direction of execution path in each time step, Linear Temporal Logic (LTL) is an immediate inadequacy for describing them. In order to describe the behavior of such systems, Sotudeh and Movaghar introduced FzCTL* in [26] such that it preserves branching time and uncertainty simultaneously. In this section, we define *Fuzzy Computation Tree Logic (FzCTL)* by restricting FzCTL*.

A temporal logic is only as powerful as the logic it is built on top of which. The more powerful the underlying logic is the more powerful the temporal logic will be in expressing model's behavior. A recent study by Haiyu Pan et al. [22] defined a more restricted Fuzzy Computation Tree Logic that only encompasses fuzzy-{negation, conjunction, disjunction, and implication} operators. Meanwhile, almost all fuzzy logic operators in FzCTL* are also included in FzCTL making it more descriptive and easier to use. Following subsection refreshes and redefines some fuzzy logic operators previously seen in [26].

3.1 Fuzzy logic operators

Fuzzy logic operators have a high variety of implementations; see [25,28]. In the present study, we use their simplest implementations whose properties are similar to that of quasi-boolean algebra related to χCTL. Fuzzy logic operations are:

$$\begin{aligned} true &= 1 \\ false &= 0 \\ \neg a &= 1 - a \\ a \sqcap b &= \min(a, b) \\ a \sqcup b &= \max(a, b) \\ a \rightarrow b &= \neg a \sqcup b = \max(1 - a, b) \end{aligned}$$

Additionally, using the saturation operator [15], shown in (1), we define *discrete-saturation* $\wr a \int_\varepsilon$ operator, please see (2).

$$\wr a \int = \max(0, \min(1, a)) \tag{1}$$

$$\wr a \int_\varepsilon = \wr \varepsilon \lfloor \frac{a}{\varepsilon} \rfloor \int \tag{2}$$

3.2 Syntax

FzCTL formulae are categorized into two types: state formulae (as φ), and path formulae (as Φ). FzCTL state formulae are formed according to the following grammar:

$$\varphi ::= r \mid x \mid \neg\varphi \mid \varphi \sqcap \varphi \mid \varphi \geq \varphi \mid \wr\varphi + \varphi\wr \mid \wr a\varphi\wr_{\varepsilon} \mid \mathrm{A}\,\Phi \mid \mathrm{E}\,\Phi$$

where $r \in [0,1]_{\Delta}$, $\varepsilon \in (0,1]_{\Delta}$, $a \in \mathbb{Q}^{+}$, and $x \in X$, that is a set of state attributes. Path formulae express temporal properties of paths; they are formed according to the following grammar:

$$\Phi ::= \mathrm{X}\,\varphi \mid \varphi\,\mathrm{U}\,\varphi$$

Although operators like *bounded-add* $\wr\varphi + \varphi\wr$, *discrete-saturation* $\wr\varphi\wr_{\varepsilon}$, and *scalar-multiplication* $\wr a\varphi\wr_{\varepsilon}$ are not seen in χCTL, they are determined by this logic. The full set of temporal operators can be defined using "Next" X and "Until" U as follows:

$$\begin{aligned}
\mathrm{AF}\,\Phi &\overset{\text{def}}{=\!=} \mathrm{A}(true\ \mathrm{U}\,\Phi)\\
\mathrm{EF}\,\Phi &\overset{\text{def}}{=\!=} \mathrm{E}(true\ \mathrm{U}\,\Phi)\\
\mathrm{AG}\,\Phi &\overset{\text{def}}{=\!=} \neg\,\mathrm{EF}\,\neg\Phi\\
\mathrm{EG}\,\Phi &\overset{\text{def}}{=\!=} \neg\,\mathrm{AF}\,\neg\Phi
\end{aligned}$$

We borrowed the definitions for *quasi-comparison* $\{\prec\succ\preceq\succeq\approx\not\approx\}$ operators from [26]. Definitions for other comparison operators are also possible through the definition of $\geq$ in conjunction with logical operators. Moreover, other auxiliary operators defined in [26] such as *bounded-subtract* $\wr\varphi - \varphi\wr$ and "if" are defined by FzCTL.

3.3 Semantic

Satisfaction relation can present truth degree of a formula. In the following notations, M represents an FzKripke model while s represents a certain state in M, and π is an infinite path defined on M.

$$\begin{aligned}
\mathbb{P}(M, s \models \varphi) &\overset{\text{def}}{=\!=} \mathbb{P}(\varphi \mid M, s)\\
\mathbb{P}(M, \pi \models \Phi) &\overset{\text{def}}{=\!=} \mathbb{P}(\Phi \mid M, \pi)
\end{aligned}$$

Semantic of state formulae is defined as follows:

$$\begin{aligned}
&\mathbb{P}(M, s \models r) = r\\
&\mathbb{P}(M, s \models x) = \mathrm{L}(s) \cdot x\\
&\mathbb{P}(M, s \models \neg\varphi) = \neg\mathbb{P}(M, s \models \varphi)\\
&\mathbb{P}(M, s \models \wr a\varphi\wr_{\varepsilon}) = \wr a\mathbb{P}(M, s \models \varphi)\wr_{\varepsilon}\\
&\mathbb{P}(M, s \models \varphi\ \ \mathrm{op}\ \ \psi) = \mathbb{P}(M, s \models \varphi)\ \ \mathrm{op}\ \ \mathbb{P}(M, s \models \psi)
\end{aligned}$$

where "op" can be a binary logical operator, a comparison operator, a quasi-comparison operator, or the bounded-add/subtract; the result of comparison operators is either "0" or "1".

$$\begin{aligned}
\mathbb{P}(M, s \models \mathrm{A}\,\Phi) &\overset{\text{def}}{=\!=} \bigsqcap_{\pi \in \mathrm{Path}_{\mathrm{inf}}(s)} \mathbb{P}(M, \pi \models_{\mathrm{A}} \Phi)\\
\mathbb{P}(M, s \models \mathrm{E}\,\Phi) &\overset{\text{def}}{=\!=} \bigsqcap_{\pi \in \mathrm{Path}_{\mathrm{inf}}(s)} \mathbb{P}(M, \pi \models_{\mathrm{E}} \Phi)
\end{aligned}$$

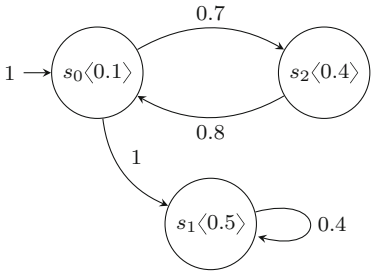


Fig. 1 Q-FzKripke K_1 with $\Delta = 0.1$ in which $X = x$ and function I returns 1 for s_0 and 0 for other states. Edges of K_1 are demonstrating the relation R while relation L is represented by decimals depicted in each state

Semantic of path formulae can be defined as follows:

$$\mathbb{P}(M, \pi \models_{\mathrm{E}} \mathrm{X}\,\varphi) \overset{\text{def}}{=\!=} \mathrm{R}(\pi[0], \pi[1]) \sqcap \mathbb{P}(M, \pi[1] \models \varphi)$$
$$\mathbb{P}(M, \pi \models_{\mathrm{A}} \mathrm{X}\,\varphi) \overset{\text{def}}{=\!=} \mathrm{R}(\pi[0], \pi[1]) \rightarrow \mathbb{P}(M, \pi[1] \models \varphi)$$
$$\mathbb{P}(M, \pi \models_{\mathrm{A}} \varphi\,\mathrm{U}\,\psi) \overset{\text{def}}{=\!=} \mathbb{P}(M, \pi \models_{\mathrm{A}} \psi) \sqcup \big(\mathbb{P}(M, \pi \models_{\mathrm{A}} \varphi) \sqcap \mathbb{P}(M, \pi \models_{\mathrm{A}} \mathrm{X}(\varphi\,\mathrm{U}\,\psi))\big)$$
$$\mathbb{P}(M, \pi \models_{\mathrm{E}} \varphi\,\mathrm{U}\,\psi) \overset{\text{def}}{=\!=} \mathbb{P}(M, \pi \models_{\mathrm{E}} \psi) \sqcup \big(\mathbb{P}(M, \pi \models_{\mathrm{E}} \varphi) \sqcap \mathbb{P}(M, \pi \models_{\mathrm{E}} \mathrm{X}(\varphi\,\mathrm{U}\,\psi))\big)$$

Given model M, satisfiability of proposition φ is defined as

$$\mathbb{P}(M \models \varphi) \overset{\text{def}}{=\!=} \bigsqcap_{s \in S} \big(\mathrm{I}(s) \rightarrow \mathbb{P}(M, s \models \varphi)\big).$$

Regarding the semantic of FzCTL formulae, the following equations are in place:

$$\mathrm{E}(\varphi\,\mathrm{U}\,\psi) \overset{\text{def}}{=\!=} \mu Z.\big(\psi \sqcup (\varphi \sqcap \mathrm{EX}\,Z)\big)$$
$$\neg\,\mathrm{A}(\varphi\,\mathrm{U}\,\psi) \overset{\text{def}}{=\!=} \nu Z.\big(\neg\psi \sqcup (\neg\varphi \sqcap \mathrm{EX}\,Z)\big)$$

where $\mu Z.f(x)$ and $\nu Z.f(x)$ are greatest and smallest fixed points of f, respectively.

Example 1 The following properties hold for K_1 as shown in Fig. 1:

$$\mathbb{P}(K_1 \models \mathrm{EF}(x)) = 0.5$$
$$\mathbb{P}(K_1 \models \mathrm{EF}(\neg x)) = 0.9$$
$$\mathbb{P}(K_1 \models \mathrm{EG}(x < 0.5)) = 0.7$$

Given an approximation error ε such that 1 is divisible by that, and an approximation function $\tau_\varepsilon(x)$, it is shown that applying $\tau_\varepsilon(x)$ on FzKripke M does not propagate the error while model checking an approximable FzCTL* proposition on it [26]. Similarly, the quantization we defined on interval [0, 1] can be regarded as an immediate approximation of FzCTL and FzKripke with approximation error $\frac{1}{\Delta}$; which obviously does not propagate the error because FzCTL is a restricted version of FzCTL* that preserves its approximability, and also due to our choice of fuzzy logic operators.

4 Quantized Fuzzy Program Graph

Fuzzy Program Graph (FzPG) provides higher-level abstraction of fuzzy systems and it that can be translated to an equivalent FzKripke model [26]. FzPG eases manual modelling process by providing a more scrutable semantics. We define Quantized Fuzzy Program Graph (Q-FzPG) quite similar to FzPG. In this section, we investigate Q-FzPG and its properties.

Definition 5 (*Quantized Fuzzy Program Graph*) This is a tuple $G = (S, s_0, X, \text{Init}, \text{Act})$ where S is a set of states, $s_0 \in S$ is the initial state, and "Init" is a function that defines degree of entrance to initial state. "Act" is a relation defining state transitions; each state transition has two parts: (1) a function that defines transition degree, and (2) a function that maps truth degrees from attribute-set of the source state to that of destination. The formal definitions for "Init" and "Act" are:

$$\begin{aligned} &\text{Init} \in \mathcal{F}_{\Delta,X} \\ &\text{Act} \in S \times S \nrightarrow \mathcal{F}_{\Delta,X} \times \mathcal{G}_{\Delta,X} \end{aligned}$$

where $\mathcal{F}_{\Delta,X} \in \mathcal{P}\big(\text{Val}_\Delta(\text{X}) \to [0,1]_\Delta\big)$ and $\mathcal{G}_{\Delta,X} = \mathcal{F}_{\Delta,X}^{|X|}$.

All functions belonging to $\mathcal{F}_{\Delta,X}$ have the constraint of being compatible with the following grammar's syntax:

$$\varphi ::= r \mid x \mid \neg\phi \mid \varphi \sqcap \varphi \mid \varphi \geq \varphi \mid \wr\varphi + \varphi\wr \mid \wr a\varphi\wr_\varepsilon$$

where $r \in [0,1]_\Delta, \varepsilon \in (0,1]_\Delta, a \in \mathbb{Q}^+$, and $x \in X$. The rest of logical operators, comparison operators, quasi-comparison, and bounded-subtract are derivable from the above defined operators.

Although "Act" is a partial function, it is possible to convert it to a total function as it was proposed for FzPG in [26]. To this end, it is enough to add missing state transitions for all nodes that are not connected directly, such that:

1. The transition possibility of the new edge is zero.
2. This edge assigns the possibility of the destination attributes by a list of zero values.

We can also make a new source by adding a dummy node ι such that "Init" continuously returns "1" for ι, meanwhile the previous "Init" function must be merged with "Act".

Example 2 If $X = (x, y)$, and $\Delta = 0.1$, then (f, g) is an edge of Q-FzPG where:

$$\begin{aligned} f(x,y) &= (x > y) \sqcap \big(0.4 \sqcup (y < 1)\big) \\ g(x,y) &= \Big\langle \big(\wr 0.5 - \wr 3x\wr_{0.2}\wr\big) \sqcap 0.9, (x > y) \sqcup \wr x + 0.2\wr_{0.1} \Big\rangle \end{aligned}$$

4.1 Semantic

In order to express the semantic of FzPG an equivalent FzKripke is inevitably required [26]; similarly there is a K_G as an equivalent Q-FzKripke to an arbitrary Q-FzPG like G such that:

1. $S' = S \times \text{Val}_\Delta(X)$
2. $\forall \eta \in \text{Val}_\Delta(X)$
   (a) if $s \neq s_0$, then $\forall s \in S \cdot \text{I}(s, \eta) = 0$
   (b) if $s = s_0$, then $\forall s \in S \cdot \text{I}(s, \eta) = \text{Init}(\eta)$

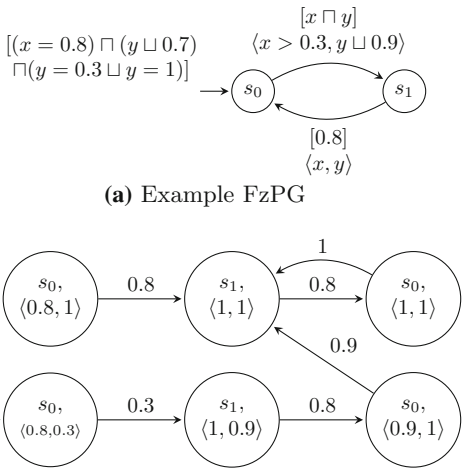


Fig. 2 An example of Q-FzPG alongside its equivalent Q-FzKripke model with $\Delta = 0.1$

3. $\forall \eta \in \mathrm{Val}_{\Delta}(X), \forall s \in S \cdot \mathrm{L}(s, \eta) = \eta$
4. $\forall \eta, \eta' \in \mathrm{Val}_{\Delta}(X), \forall s, s' \in S$
   (a) if $\mathrm{Act}(s, s')$ is not defined, then $\mathrm{R}((s, \eta), (s', \eta')) = 0$
   (b) if $(\mathrm{A}, \mathrm{B}) = \mathrm{Act}(s, s')$, and $\eta' = \mathrm{B}(\eta)$ then $\mathrm{R}((s, \eta), (s', \eta')) = \mathrm{A}(\eta)$

When the equivalent Q-FzKripke accepts a proposition with a certain truth degree, we say that initial Q-FzPG also accepts the proposition with the same degree. Although there is an equivalent Q-FzPG denoted by G_K with the same number of states $|S|$ to an arbitrary Q-FzKripke like K, it is against the purpose of Fuzzy Program Graph, which is to model the system in a highly compressed format, because for each state of G_K there is an equivalent state in K. To exemplify, an FzPG and an equivalent FzKripke are depicted in Fig. 2.

From this point onwards, for all Q-FzPGs we assume the "Init" function returns "1" for initial state and "0" for all other states unless otherwise stated. In fact, we define a novel "Init_{ι}" function that returns "1" for a dummy source like ι; thereupon, by merging "Init" and "Act", we obtain a Single-Source Q-FzPG that is referred to instead of the original Q-FzPG.

5 Symbolic checking of FzCTL on FzPG

Heretofore several symbolic methods are proposed to investigate satisfiability of propositions of multi-valued temporal logics, some of which use *ordered binary decision diagram* (OBDD), others use vectors of similar diagrams to store multi-valued Kripke models and investigate the satisfiability of propositions on that model [6,7].

In order to check the temporal properties of a particular FzPG, initially, we translate it to an equivalent FzKripke; then, we check temporal properties on that FzKripke. This yields to an exponential blow-up in the state-space; thus, it is more appropriate to check the specification directly over the FzPG. To this purpose, we devised a symbolic checking method using OBDDs and vectors of OBDD.

Let $\Delta = 2^{-d}$, then since all numbers involved in model construction and model checking are multiples of Δ, a data type with $d + 1$ bits is sufficient to store truth degrees, where the first bit of data type is the least significant bit and the d^{th} bit is the most significant one. In

order to store a truth degree like ε in $d+1$ bits, it is enough to store $\varepsilon \times 2^d$ instead. For further readability, from this point forward we assume $d' = d + 1$.

Example 3 Let $\Delta = 4$ then we have:

$$1.00 \equiv 1.00 \times 2^4 = (10000)_2$$
$$0.50 \equiv 0.50 \times 2^4 = (01000)_2$$
$$0.\overline{33} \equiv 0.\overline{33} \times 2^4 = (00101)_2$$
$$0.25 \equiv 0.25 \times 2^4 = (00100)_2$$
$$0.19 \equiv 0.19 \times 2^4 = (00011)_2$$
$$0.00 \equiv 0.00 \times 2^4 = (00000)_2$$

We also encode attributes in a symbolic form; suppose W is a vector of bit vectors representing the set of all attributes X as follows:

$$W = \langle W_1, \ldots, W_k \rangle, \quad k = |X|$$

where W_i is a bit vector with length d' representing encoded attribute x_i, hence W_{ij} is a bit in vector W_i.

$$W_i = \langle W_{i0}, \ldots, W_{id} \rangle, \quad i \in 1 \cdots k$$

Let U be union of variables used to encode model states S; this union has a total number of $\lceil \log_2 |S| \rceil$ members. From this point forward, we repeatedly use copies of U and W denoted by U' and W' respectively.

5.1 Auxiliary OBDD operators

This subsection is devoted to describe the functionality of important vector operations, besides basic operations defined on OBDDs that are necessary to define the proposed method.

Definition 6 Let v be a vector in terms of variables W and U, the OBDD of vector v is of maximum size 2^h and maximum height h as follows:

$$h = |W| + |U| = n + kd' \tag{3}$$

where $n = |U|$, $k = |X|$ and d' is the length of W.

Theorem 1 ([16]) *Logical combination of two OBDDs of size m and n is of time* $O(mn)$.

5.1.1 vConst *and* vVal

Converting integers and constants to vectors and viceversa is essential. Given an integer like "c" its vector representation is returned by function vConst. On the other side, function vVal returns integer representation of a vector like "v". We define constants "True" and "False" using vConst as follows:

$$\text{False} \stackrel{\text{def}}{=\!=} \text{vConst}(0)$$
$$\text{True} \stackrel{\text{def}}{=\!=} \text{vConst}(2^d)$$

Similarly, in order to convert a truth degree like $r \in [0, 1]_\Delta$, we have:

$$\text{Const}(r) \stackrel{\text{def}}{=\!=} \text{vConst}(r \times 2^d).$$

Algorithm 1

```
procedure vGEQ(v1[0 ... d], v2[0 ... d])
    if d > 0 then
        return (v1[d] > v2[d] ∨ v1[d] = v2[d]) ∧ vGEQ(v1[0 ... d − 1], v2[0 ... d − 1])
    end if
    return v1[0] > v2[0] ∨ v1[0] = v2[0]
end procedure
```

5.1.2 vIF

When called like $\text{vIF}(x, v_1, v_2)$, this returns an OBDD vector whose i^{th} element equates to diagram $\text{if}(x, v_1[i], v_2[i])$. Parameter x is an OBDD, v_1 and v_2 are two vectors with d' diagrams. vIF calculates the result iteratively in time $O(d')$.

5.1.3 Comparison

Comparison operators return an OBDD by comparing two OBDD vectors. These are defined recursively (e.g. Algorithm 1). Operators "$=, \neq, >, \geq, <, \leq$" and function prototypes "vEQL, vNEQ, vGRT, vGEQ, vLST, vLEQ", are used interchangeably (e.g. $\text{vLEQ}(v_1, v_2) \equiv v_1 \leq v_2$). Comparing two OBDD vectors of length d' is convertible to an iterative logical combination of two OBDDs and the result is an OBDD of height h. The iteration is of order d' thus the computational complexity of a comparison operator is $O(d'2^{2h})$

5.1.4 vADD *and* vSUB

Operators vADD and vSUB perform element-wise addition and subtraction of two vectors with similar length, the result is obviously a vector of same length. vADD and vSUB are also convertible to an iteration of order d' of logical combination of OBDDs; therefore, they also run in time $O(d'2^{2h})$. Similarly, bounded-add and bounded-subtract are permissible by comparison operations and vIF; this indicates that they are of the same order as $O(d'2^{2h})$.

5.1.5 Logical connectives

Logical operations over OBDD vectors are obtainable via previously defined operators. All logical operators run in time $O(d'2^{2h})$.

$$\text{NOT}(v_1) \stackrel{\text{def}}{=\!=} \text{vSUB}(\text{TRUE}, v_1) \tag{4}$$

$$\text{OR}(v_1, v_2) \stackrel{\text{def}}{=\!=} \text{vIF}(\text{vGEQ}(v_1, v_2), v_1, v_2) \tag{5}$$

$$ss\text{AND}(v_1, v_2) \stackrel{\text{def}}{=\!=} \text{vIF}(\text{vGEQ}(v_1, v_2), v_2, v_1) \tag{6}$$

Conjunction $\wedge$ and disjunction $\vee$ operators, and function prototypes AND and OR are interchangeably used from this point onward.

5.1.6 vMUL *and* vDIV

Operators vMUL (vDIV) multiplies (divides) elements of a vector v_1 by a constant factor like c. The result is obviously a vector of the same length.

5.1.7 ADD *and* MUL

We define add or multiply operations over two vectors as follows:

$$\begin{aligned}
&\textsc{Add}(v_1, v_2) \overset{\text{def}}{=\!=} \textsc{vIf}(\textsc{vGeq}(v_1, \textsc{Not}(v_2)), \textsc{True}, \textsc{vAdd}(v_1, v_2))\\
&\textsc{Mul}(v_1, p, q, h) \overset{\text{def}}{=\!=} \textsc{vMul}(\textsc{vDiv}(\textsc{vMul}(v_1, p), qh), h)
\end{aligned}$$

where $h = \frac{\varepsilon}{\Delta}$, $a = \frac{p}{q}$ and $p, q \in \mathbb{N}$.

5.2 Compiling FzCTL property to OBDD vector

For each sub-formula of FzCTL like φ, an OBDD vector is calculated recursively called $\tau(\varphi)$ using above defined auxiliary OBDD functions. Following are recursive procedures to calculate τ for numbers, attributes and operators of a formula.

$$\begin{aligned}
&\tau(r) \overset{\text{def}}{=\!=} \textsc{Const}(r)\\
&\tau(x_i) \overset{\text{def}}{=\!=} W_i\\
&\tau(!\varphi) \overset{\text{def}}{=\!=} \textsc{Not}(\tau(\varphi))\\
&\tau(\varphi \sqcap \psi) \overset{\text{def}}{=\!=} \textsc{And}(\tau(\varphi), \tau(\psi))\\
&\tau(\varphi \geq \psi) \overset{\text{def}}{=\!=} \textsc{vGeq}(\tau(\varphi), \tau(\psi))\\
&\tau(\mathrm{EX}\,\varphi) \overset{\text{def}}{=\!=} \textsc{Ex}(\tau(\varphi))\\
&\tau(\mathrm{AX}\,\varphi) \overset{\text{def}}{=\!=} \textsc{Ax}(\tau(\varphi))\\
&\tau(\mathrm{EU}(\varphi, \psi)) \overset{\text{def}}{=\!=} \textsc{Eu}(\tau(\varphi), \tau(\psi))\\
&\tau(\mathrm{AU}(\varphi, \psi)) \overset{\text{def}}{=\!=} \textsc{Au}(\tau(\varphi), \tau(\psi))\\
&\tau\big(\wr\varphi + \psi\wr\big) \overset{\text{def}}{=\!=} \textsc{Add}(\tau(\varphi), \tau(\psi))\\
&\tau\big(\wr a\varphi\wr_\varepsilon\big) \overset{\text{def}}{=\!=} \begin{cases} \textsc{Mul}(\tau(\varphi), p, q, h) & a \leq 1\\ \textsc{vIf}(\textsc{vGeq}(\tau(\varphi), \textsc{Const}(1/a)), & \\ \quad \textsc{True}, \textsc{Mul}(\tau(\varphi), p, q, h)) & a > 1 \end{cases}
\end{aligned}$$

where $h = \frac{\varepsilon}{\Delta}$, $a = \frac{p}{q}$ and $p, q \in \mathbb{N}$. We can simply define the truth degree of proposition φ, over a model like K_G as follows:

$$\mathbb{P}(K_G \models \varphi) \overset{\text{def}}{=\!=} \bigsqcap_{\eta \in \mathrm{Val}(X)} (\mathbb{P}(M, (s, \eta) \models \varphi)) = \bigsqcap_{\eta \in \mathrm{Val}(X)} \theta(s, \eta).$$

where $\theta(s, \eta)$ denotes $\mathbb{P}(M, (s, \eta) \models \varphi)$ and $\tau(\varphi)$ is implicitly represented by vector θ. Conjunction operator $\bigsqcap$ is obtainable using following property:

Property 1 *For conjunction and disjunction of all member of set* $\mathrm{I} \subset \mathbb{R}$ *we have:*

$$\bigsqcap_{i\in\mathrm{I}} a_i = D \Leftrightarrow \exists i \in \mathrm{I} \cdot a_i = D \wedge \forall i \in \mathrm{I} \cdot a_i \geq D \Leftrightarrow \exists i \in \mathrm{I} \cdot a_i \leq D \wedge \forall i \in \mathrm{I} \cdot a_i \geq D$$

$$\bigsqcup_{i\in\mathrm{I}} a_i = D \Leftrightarrow \exists i \in \mathrm{I} \cdot a_i = D \wedge \forall i \in \mathrm{I} \cdot a_i \leq D \Leftrightarrow \exists i \in \mathrm{I} \cdot a_i \geq D \wedge \forall i \in \mathrm{I} \cdot a_i \leq D$$

From this point onward, we refer to D as a vector of variables like $D = \langle z_d, \ldots, z_0 \rangle$. Following algorithm is the required procedure to convert OBDD vector $\tau(\varphi)$, which is represented by $\theta(U, W)$, into a decimal number to determine the truth degree of proposition φ over an FzPG. Depending on the algorithms certain preprocesses may be involved prior to evaluate truth degree of $\tau(\varphi)$, some of which are referred later in this paper.

Algorithm 2

1: **procedure** EVALUATE($\theta(U, W)$)
2: $\alpha := U =$ VCONST(s_0)
3: **for** i := 0 to d **do**
4: $\lambda[i] := \theta[i] \wedge \alpha$
5: **end for**
6: $\beta_1 := \exists U \cdot \exists W \cdot \lambda = D$
7: $\beta_2 := \forall U \cdot \forall W \cdot \lambda \geq D$
8: $\beta := \beta_1 \wedge \beta_2$
9: **for** i := 0 to d **do**
10: $\rho[i] := \exists D \cdot (D[i] \wedge \beta)$
11: **end for**
12: return $\frac{\text{VVAL}(\rho)}{2^d}$
13: **end procedure**

In line 2, we restrict members of vector α to node s_0 in time $O(d'2^h)$, then we restrict members of θ to nodes in α (that is s_0) in line 4 to create vector λ in time $O(d'2^h)$. In lines 6 and 7, the comparison operators are of time complexity of $O(d'2^{2(h+d')})$ and results are of memory complexity of $O(2^{h+d'})$. In those lines, the universal and existential quantifiers are restriction operators each of which is of time complexity of $O(2^{h+d'})$. These quantifiers eliminate participant variables in diagrams and reduce the height of corresponding diagrams gradually by one. Conjunction over β_1 and β_2 forms β in time $O(2^{2d'})$.

According to properties of $\bigsqcap$, β is a composition of minterms of variables z_0 to z_d. In line 10, expression $D[i]$ represents z_i, so when z_i participates in β, expression $\rho[i]$ evaluates to 1 otherwise to 0, the complexity of calculating vector ρ is $O(2^{d'})$. Finally, dividing VVAL(ρ) by 2^d results in the truth degree of φ. This algorithm is of time complexity of $O(d'2^{2(h+d')})$.

5.3 Temporal operators

An implication operator represents the transition edge between nodes s and t. A disjunction over all transition implications forms relation R. In order to add a transition edge to relation R, we initially calculate and store vectors of Act(s, t), then using implication operator we calculate the transition edge that eventually aggregates to R with a disjunction; please see Algorithm 3.

In line 2, it is assumed the Act(s, t) is a total function. Otherwise, we repeatedly call ADDRELATION on each and every one of transition edges between each pair. Since W' and B

Algorithm 3

1: **procedure** ADDRELATION(s, t)
2: $(A, B) := \mathrm{Act}(s, t)$
3: $\alpha := U(s) \wedge U'(t) \wedge (W' = B)$
4: $\mathrm{R} := \mathrm{R} \vee \mathrm{VIF}(\alpha, A, \mathrm{FALSE})$
5: **end procedure**

are both vectors of k rows and each row is consist of d' elements, OBDDs for these variables is of height kd'. Accordingly, $\mathrm{VEQL}(W'_i, B_i)$ is of complexity of $O(d'2^{2kd'})$ and therefore $\mathrm{VEQL}(W', B)$ is of time complexity of $O(kd'2^{2kd'})$. Line 3, calculates α in time $O(2^{2h+1})$ and line 4 runs in time $O(d'2^{2h+1})$; subsequently, calculating relation R is of the following complexity:

$$O\big(r(2^{2h+1} + d'2^{2h+1} + kd'2^{2kd'})\big) = O\Big(r\big(2^{2h+1}(1+d') + kd'2^{2kd'}\big)\Big)$$
$$= O\Big(r\big(d'2^{2h+1} + kd'2^{2kd'}\big)\Big) = O\Big(rd'2^{2kd'}\big(2^{2n+1} + k\big)\Big).$$

5.3.1 EX operator

According to semantic of EX operator in FzCTL over K_G model, we have:

$$\mathbb{P}(M, (s, \eta) \models \mathrm{EX}\,\varphi) = \bigsqcup_{(s',\eta')\in S'} \mathbb{P}(M, (s', \eta') \models \varphi) \sqcap \mathrm{R}((s, \eta), (s', \eta')). \tag{7}$$

where R can be obtained according to its definition in K_G. Now we can rewrite (7) by substituting $\tau(\varphi)$ with θ and $\tau(\mathrm{EX}\,\varphi)$ with ρ as follows:

$$\rho(s, \eta) = \bigsqcup_{(s',\eta')\in S'} \theta(s', \eta') \sqcap \mathrm{R}((s, \eta), (s', \eta')).$$

The simplest method to calculate $\rho(s, \eta)$ is to find the maximum value for conjunction of θ and vector R [7]. Algorithm 4 illustrates a procedure to calculate ρ. In this algorithm, ψ represents the equality of vector D and vector ρ (i.e., a vector of diagrams in terms of U and W) as follows:

$$\psi \overset{\text{def}}{=\!=} \big(\langle\rho_d, \ldots, \rho_0\rangle = \langle z_d, \ldots, z_0\rangle\big) \overset{\text{def}}{=\!=} \rho_d \oplus z_d \wedge \cdots \wedge \rho_0 \oplus z_0$$

where $\oplus$ represents eXclusive-OR. In line 8 of algorithm 4, the expression $\exists D \cdot (D[i] \wedge \psi)$ forms $\rho[i]$, as for $D' = \langle z_d, \ldots, z_{i+1}, z_{i-1}, \ldots, z_0\rangle$ we have:

$$\exists D \cdot (z_i \wedge \psi) = \exists D' \cdot \exists z_i \cdot (z_i \wedge \psi) = \exists D' \cdot \exists z_i \cdot \Big(z_i \wedge \rho_i \wedge \big(\bigwedge_{i\neq j} z_j \oplus \rho_j\big)\Big)$$
$$= \exists D' \cdot \Big(\rho_i \wedge \big(\bigwedge_{i\neq j} z_j \oplus \rho_j\big)\Big) = \rho_i \wedge \exists D' \cdot \Big(\bigwedge_{i\neq j} z_j \oplus \rho_j\Big)$$
$$= \rho_i \wedge 1 = \rho_i$$

Bits in θ are of height h and bits of R are of height $2h$. Line 2 forms θ' from θ in time $O(d'2^h)$. In line 3, λ is formed by applying AND on two vectors of d' bits which according to (4) is convertible to VGEQ that takes $O(d')$ times of conjunctive and disjunctive operations on corresponding bits. Composition of individual bits of θ' and R is of order $O(2^{3h})$ and the resulted bit is of height $2h$. Conjunctive and disjunctive operations on diagrams of height

Algorithm 4 $\rho(U, W) = \text{EX}\big(\theta(U, W)\big)$

1: **procedure** $\rho(U, W)$
2: $\quad\theta' := \theta[U'/U, W'/W]$
3: $\quad\lambda := \theta' \wedge \text{R}$
4: $\quad\psi_1 := \exists U' \cdot \exists W' \cdot \lambda \geq D$
5: $\quad\psi_2 := \forall U' \cdot \forall W' \cdot \lambda \leq D$
6: $\quad\psi := \psi_1 \wedge \psi_2$
7: $\quad$**for** $i := 0 \ldots d$ **do**
8: $\quad\quad\rho[i] := \exists D \cdot (D[i] \wedge \psi)$
9: $\quad$**end for**
10: **end procedure**

$2h$ is of order $O(2^{4h})$, which leads us to the computational complexity of $O(d'2^{4h})$ and a result of height $2h$ for VGEQ. The VIF is of the same computational complexity therefore total complexity of line 3 is $O(d'2^{4h})$.

In order to form ψ_1 and ψ_2, comparing vectors of size d' bits with diagrams of height d' and h is required as shown in lines 4 and 5. These comparison operations result in diagrams of maximum height $d' + h$ in time complexity of $O(d'2^{2(h+d')})$; through a similar reasoning that we used while computing λ in line 3. Universal and existential quantifiers in lines 4 and 5eliminate participant variables in diagrams and reduce the height of corresponding diagrams gradually by one, consequently all predicates in these lines (i.e., $\lambda \geq D$ and $\lambda \leq D$) are quantified in time complexity of $O(2^{2h+d'})$, because we have:

$$O\big(2^{2h+d'} + 2^{2h+d'-1} + \cdots + 2^{h+d'}\big) = O\big(2^{2h+d'}\big).$$

Time complexity for universal and existential quantification is negligible in comparison with the time complexity of comparison operators therefore lines 4 and 5are of time complexity $O(d'2^{2(h+d')})$. Since ψ_1 and ψ_2 are of maximum height of $h + d'$ the their conjunction in line 6 is of time $O(2^{2(h+d')})$. Subsequently, ψ is obtainable in total time $O(d'2^{2(h+d')})$. In line 8, $D[i] \wedge \psi$ is of time $O(2^{h+d'})$; moreover, computational complexity of existential quantification, $\exists D$, is as follows:

$$O\big(2^{h+d'} + 2^{h+d'-1} + \cdots + 2^{h+1}\big) = O\big(2^{h+d'}\big).$$

Having computed ψ, then ρ is obtainable in time $O(d'2^{2(h+d')})$; therefore, the total complexity for EX operator is as follows:

$$O\big(d'2^{4h} + d'2^{2(h+d')} + 2^{h+d'}\big) = O\big(d'2^{4h}\big).$$

5.3.2 EU operator

The semantic of $\text{E}(\varphi\,\text{U}\,\psi)$ indicates it only equates to "1" only if ψ eventually occurs. We formally define "strong until" using the fixed-point concept as shown in Algorithm 5. Computing EU requires EX and a variety of auxiliary operations to be called repetitively. For each iteration computational complexity of auxiliary operators are negligible comparing to that of EX operator. EU operator give rise to a sequence of vectors as follows:

$$Z_1, Z_2, \ldots, Z_g$$

Algorithm 5

```
1: procedure EU(v1, v2)
2:     Z := v2
3:     while TRUE do
4:         L := v2 ∨ (v1 ∧ EX(Z))
5:         if VEQL(L, Z) = 1 then
6:             return Z
7:         end if
8:         Z := L
9:     end while
10: end procedure
```

where Z_i is equivalent to a function that maps members of S' to $[0, 1]_\Delta$. This equivalency can be demonstrated as follows:

$$Z_i = \left\{\left((s_1, \eta_1), v_1^{(i)}\right), \ldots, \left((s_e, \eta_e), v_e^{(i)}\right)\right\} \quad \forall j \in \{1 \ldots e\} \cdot v_j^{(i)} \in [0, 1]_\Delta$$

where $e = |S'|$. In each iteration of EU some values of $v_j^{(i)}$ are increased till convergence happens in the g^{th} iteration. This property can be formalized as:

$$\forall i \in 1 \ldots (g-1) \cdot \left(\forall j \in 1 \ldots e \cdot v_j^{(i+1)} \geq v_j^{(i)} \wedge \exists j \in 1 \ldots e \cdot v_j^{(i+1)} \geq v_j^{(i)} + \Delta\right) \quad (8)$$

According to (8) it is obvious $g \leq \frac{e}{\Delta}$, because in each iteration there exists at least one v_j whose value is incremented by a minimum of Δ. In order to store all members of S' we use all variables U and W, consequently $e = O(2^h)$ and $g = O(2^{h+d'})$. In practice the convergence happens quickly and this is the worst-case convergence time. Given the computational complexity of EX operator, the computational complexity of EU is of $O(d'2^{5h+d'})$.

5.3.3 AU operator

The definition of AU operator is quite similar to EU and similar reasoning applies to its time complexity analysis; please see Algorithm 6.

Algorithm 6

```
1: procedure AU(v1, v2)
2:     Z := NOT(v2)
3:     while TRUE do
4:         L := NOT(v2) ∧ (NOT(v1) ∨ EX(Z))
5:         if VEQL(L, Z) = 1 then
6:             return NOT(Z)
7:         end if
8:         Z := L
9:     end while
10: end procedure
```

5.3.4 AX operator

The definition of AX is directly obtained from the definition of EX operator as shown in Algorithm 7, therefore the time complexity of this operator is similar to that of EX operator.

Algorithm 7

1: **procedure** AX(v_1)
2: return NOT(E X(NOT(v_1)))
3: **end procedure**

5.4 Computational complexity analysis

Forming relation R is the preprocessing step of checking proposition φ over a model, then operators of temporal logic are computed recursively. Finally, we evaluate the truth degree of φ in a post-processing step. The most costly and time-consuming stage in verification of proposition φ are EU and AU operators. Let $|\varphi|$ be the length of proposition φ, then for overall time complexity we have:

$$\begin{aligned} &O(rd'2^{2kd'}(k+2^{2n})+|\varphi|\times(d'2^{5h+d'})+(d'2^{2(h+d')})) \\ &\quad = O(d'2^{2h}(r(1+k2^{-2n})+|\varphi|\times 2^{3h+d'}+2^{2d'})) \end{aligned}$$

since k is naturally a small number and $r = O(|S|^2) = O(2^{2n})$ then computational complexity of checking φ over a model is of computational complexity of $O(|\varphi| \times d'2^{5h+d'})$.

Please note that we are counting computational steps with different scaling parameters. Unlike the general convention of considering computational complexity by the size of state-space, we considered the computational complexity by the length of encoding. The reason for this unorthodox convention is simply that our applications of interest required a speed-accuracy trade-off. In other words, we simply calculated the willingness to respond slowly with fewer errors compared to quick response with relatively more errors.

Whilst this manuscript was under review authors were informed that a similar study is done for model checking of a fuzzy logic extended CTL over a fuzzy logic extended Kripke Structure [22]. They claimed the computational complexity of checking fuzzy logic extended CTL formulae φ over a fuzzy logic extended Kripke structure is as follows:

$$O(|\varphi| \times (|S'|^2 + |S'| \times \log|S'|)),$$

where S' is the state-space of the Kripke structure. Please note, the state-space of an FzKripke structure can be larger than that of an equal FzPG by an exponential factor; that is, $O(|S'|) = O(|S \times \mathrm{Val}_{\Delta}(X)|)$, where S is the state-space of the FzPG. The computational complexity of our model checking approach with respect to state-space of equal FzKripke is S' is of $O(|S'|^5)$, which is by no means tight.

Nevertheless, in order to model check a given FzPG, we need to construct the complete state-space of an equal FzKripke including its whole transition edges so that we are able to model check the equal FzKripke structure using the approach in [22]. This is computationally expensive and eventually leads to an state-space explosion; yet, since our approach directly model checks the FzCTL formula on the FzPG, state space explosion is effectively avoided; Sects. 6.1 and Appendix A.2 provide intuitive assessments of our method by reporting practical execution time and memory consumption.

The memory complexity of model checking an FzKripke structure in both approaches is of $O(|S'|^2)$. Although simple operations like comparison are of computational complexity of $O(d)$ while using OBDD, we believe OBDD provides a more compact and more efficient representation of our state-space. Further research is necessary to provide an insightful comparison of these two approaches in terms of expressiveness, applicability, and scalability.

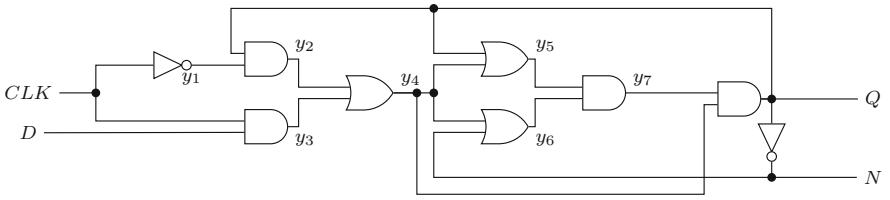


Fig. 3 Block diagram of multi-valued D flip-flop that is proposed in [8]. AND gates represent min function; similarly OR gates represent max function. We have labeled the output of each and every gate with y_1 to y_7, N, and Q for further readability

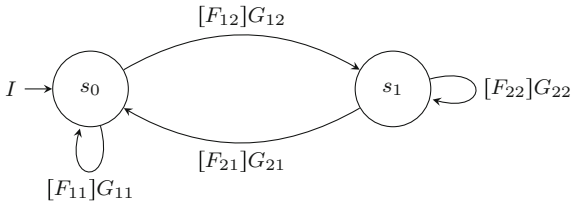


Fig. 4 FzPG G which represents the multi-valued D flip-flop depicted in Fig. 3

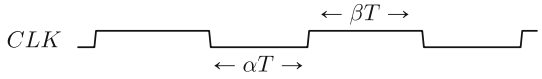
6 Fuzzy flip-flops

The concept of fuzzy flip-flops was first proposed by Hirota et al. and then studied in [8,13,14,19,20]. They presented the idea of designing fuzzy logic hardware systems using fuzzy flip-flops. Fuzzy flip-flops are made of fuzzy logic gates through having some extensions to binary flip-flops. Almost none of fuzzy flip-flops are suitable for realizing neurons in a multilayer perceptron because instability is the issue with almost all of them; this means the output of the circuit may fluctuate under certain conditions. In this section, we first investigate the correctness of a previously proposed multi-valued D flip-flop using our proposed formal method, and then we introduce a formal model of fuzzy J-K flip-flop and investigate the correctness of its behavior.

6.1 Fuzzy D Flip-Flop

Ben Choi and Kankana Shulka proposed a multi-valued D flip-flop in [8]. Although they investigated the validity of their proposed circuit via computer simulations, under certain conditions gate delays lead to dynamic hazards. In this section, we show under which condition an unpredictable sequence of outputs are generated periodically for Choi's circuitry. Another multi-valued D flip-flop is verified with a quite similar method, which led to discovery of static hazards in the circuitry [26].

Figure 3 illustrates the block diagram of Choi's D flip-flop. For the sake of simplicity, we assume propagation delay for all gates are the same and the value is equal to $T = 2^{-p}$ where p is a positive integer. We also assume input D is stable and the CLK is a pulse in the following form:



where α and β are integers. Great values of α and β do not cause any problems except slowing down the circuit. On the other hand, if α and β would be smaller than a limit there are no chance for the circuit to stabilize its output therefore it acts incorrectly. We can prove

by model checking that this design requires the values of α and β to be at least 7. Let us assume $N = \frac{1}{T}$ be the maximum possible value for α and β (obviously this is an assumption to simplify the modeling problem, greater values would not affect the circuit but slowing it down).

We have also labeled all the outputs for each logical gate in this block diagram, please see Fig. 3. Fresh primed labels are used to denote the outputs of logical gates after T time units; now it is easy to compute the output of each gate considering its inputs. From this point onward we denote *CLK* as C for further readability.

$$
\begin{aligned}
y_1' &= \neg C \\
y_2' &= y_1 \sqcap Q \\
y_3' &= D \sqcap C \\
y_4' &= y_2 \sqcup y_3 \\
y_5' &= y_4 \sqcup Q \\
y_6' &= y_4 \sqcup N \\
y_7' &= y_5 \sqcap y_6 \\
Q' &= y_4 \sqcap y_7 \\
N' &= \neg Q
\end{aligned}
$$

Now we consider an FzPG with two states s_0 and s_1. Being in state s_0 means *CLK* is 0 and being in S_1 conveys the opposite, please see Fig. 4. The attribute set X is as follows:

$$X = \langle \Gamma, u, D, C, y_1, y_2, y_3, y_4, y_5, y_6, y_7, Q, N \rangle$$

Attribute Γ represents the passage of time in states of FzPG with steps of T, therefore on the verge of entering a new state the value for this attribute is 0; whilst on a state, as soon as Γ changes the output of all gates will change accordingly. Attribute u represents the raising edge of the clock pulse; it is set to 1 while the clock is pulse raised. Once u is set it will preserve its value. Participating functions in graph G are defined as:

$$
\begin{aligned}
I &= (\Gamma = 0) \sqcap (C = 0) \sqcap (u = 0) \\
F_{11} &= (\Gamma < \alpha T) \\
F_{12} &= (\Gamma = \alpha T) \\
F_{22} &= (\Gamma < \beta T) \\
F_{21} &= (\Gamma = \beta T) \\
G_{11} = G_{22} &= \langle \lfloor \Gamma + T \rfloor, u, D, C, \neg C, y_1 \sqcap Q, D \sqcap C, \\
&\qquad y_2 \sqcup y_3, y_4 \sqcup Q, y_4 \sqcup N, y_5 \sqcap y_6, y_4 \sqcap y_7, \neg Q \rangle \\
G_{12} &= \langle 0, 1, D, 1, y_1, y_2, y_3, y_4, y_5, y_6, y_7, Q, N \rangle \\
G_{21} &= \langle 0, u, D, 0, y_1, y_2, y_3, y_4, y_5, y_6, y_7, Q, N \rangle
\end{aligned}
$$

6.1.1 Properties of multi-valued D flip-flop

In order to verify Choi's design of multi-valued D flip-flop, we defined its properties using FzCTL and then investigated their correctness using model checking.

Table 1 Unstable condition of fuzzy D flip-flop

Time	C	y_1	y_2	y_3	y_4	y_5	y_6	y_7	Q	N
t	1	0	0	D	D	D	$\overline{D} \sqcup D$	D	D	$\overline{D}$
$t+T$	0	1	0	0	D	D	$\overline{D} \sqcup D$	D	D	$\overline{D}$
$t+2T$	0	1	D	0	0	D	$\overline{D} \sqcup D$	D	D	$\overline{D}$
$t+3T$	0	1	D	0	D	D	$\overline{D}$	D	0	$\overline{D}$
$t+4T$	0	1	0	0	D	D	$\overline{D} \sqcup D$	$\overline{D} \sqcap D$	D	1
$t+5T$	0	1	D	0	0	D	1	D	$\overline{D} \sqcap D$	$\overline{D}$
$t+6T$	0	1	$\overline{D} \sqcap D$	0	D	$\overline{D} \sqcap D$	$\overline{D}$	D	0	$\overline{D} \sqcup D$
$t+7T$	0	1	0	0	$\overline{D} \sqcap D$	D	$\overline{D} \sqcup D$	$\overline{D} \sqcap D$	0	1
$t+8T$	0	1	D	0	0	D	1	D	$\overline{D} \sqcap D$	$\overline{D}$
$t+9T$	0	1	$\overline{D} \sqcap D$	0	D	$\overline{D} \sqcap D$	$\overline{D}$	D	0	$\overline{D} \sqcup D$
$t+10T$	0	1	0	0	$\overline{D} \sqcap D$	D	$\overline{D} \sqcup D$	$\overline{D} \sqcap D$	D	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Property 2 *$6T$ after the very first raising edge of clock pulse, we will have $Q = D$ for the rest of time. This property is expressible in FzCTL as follows:*

$$\mathrm{AG}\left(u = 1 \rightarrow \mathrm{AX}^6\left(\mathrm{AG}(Q = D)\right)\right)$$

where D is input, AX^n denotes applying AX *operator n consecutive times. This proposition evaluates to* 0.

Property 3 *$6T$ after the very first raising edge of clock pulse, the possible values for Q are D, $\overline{D}$, and* 0. *This property is expressible in FzCTL as follows:*

$$\mathrm{AG}\left(u = 1 \rightarrow \mathrm{AX}^6\left(\mathrm{AG}(Q = D \sqcup Q = 0 \sqcup Q = \overline{D})\right)\right)$$

Finally, this proposition evaluates to 1.

Table 1 depicts a trace that nullifies the first property. There is an issue caused by the propagation delay of NOT gate while falling edge occurs. According to Table 1, a dynamic hazard with a period of $3T$ emerged at time $t + 5T$. As can be observed if $D < \overline{D}$ then on the verge of clock's falling edge the next value for Q is among D, 0, or $\overline{D}$ otherwise if $D > \overline{D}$ then Q is either D or 0. A short while after clock's rising edge (i.e. less than $6T$) the state of circuit will be reverted to column t. As long as clock pulse is high, the state is preserved.

6.1.2 Experimental results

We ran the experiments on a laptop with 4GBytes of RAM with Intel® Core™2 Due (2.6 G.Hz.) CPU that runs Windows XP Professional (32bits). We used BuDDy library [16] to implement proposed algorithms because this library provides a near comprehensive set of tools and functions to work with OBDDs and their vectors. In this library nodes of all diagrams are stored and retrieved by a hashing method. The total size allocated for each node is 20 bytes and the upper bound for number of nodes is initialized by function "bdd_init" which we set it to 100,000,000; that is, a total memory of 2GBytes. Due to memory space

Table 2 Given $h = 5$ and considering different values of α and β, we recorded number of calls to Ex operator (#Ex), execution time in milliseconds, and memory consumption in terms of BDD nodes

α	β	#Ex	Time (ms)	#Nodes
8	8	23	918	632,944
16	8	31	1639	1,040,096
8	16	31	1954	1,171,231
16	16	39	2441	1,314,031
24	8	39	2673	1,436,208
8	24	39	3046	1,613,500
32	8	47	3832	1,742,449
16	24	47	3924	1,882,932
24	16	47	3974	1,838,548
24	24	55	4645	1,969,160
32	16	55	5854	2,183,117
8	32	47	6365	2,090,664
32	24	63	7121	2,457,824
16	32	55	8082	2,390,179
24	32	63	10,430	2,622,073
32	32	71	13,006	2,652,580

Table 3 Given $\alpha T = \beta T = \frac{1}{2}$ we recorded the state-space size, number of calls to Ex operator (#Ex), execution time in milliseconds, and memory consumption in terms of BDD nodes for different values of h

h	State-space	#Ex	Time (ms)	#Nodes
4	2^{35}	23	660	526,395
5	2^{42}	39	2441	1,314,031
6	2^{49}	71	21,206	2,821,965
7	2^{56}	135	129,730	5,715,114
8	2^{63}	263	431,222	11,358,475
9	2^{70}	519	1,530,087	22,485,129
	$O(2^{7h+7})$	$O(2^h + 7)$	$O(2^{1.85h+4})$	$O(2^{h+15.44})$

constraints, as soon as 70% of the allocated memory is consumed, we reorder diagrams to reduce number of nodes as well as consumed memory.

We used different parameter values for α and β ranging from 2^3 to 2^5; please see Table 2. For all cases the first property evaluates to 0 while the second property evaluates to 1. In case of parameter values less than 7 for α or β both properties evaluate to 0. Changing T to any value ranging from 2^{-6} to 2^{-3} did not affect the results yet it affected memory consumption and execution time of the proposed model checker. Meanwhile, Table 3 presents the practical orders on computational power and memory space needed for our proposed method. As can be seen by reported computational and memory complexity are not tight.

6.2 Fuzzy J-K flip-flop

In this subsection we are destined neither to provide an on-chip fuzzy system nor their applications but a formal model realizing a fuzzy flip-flops.

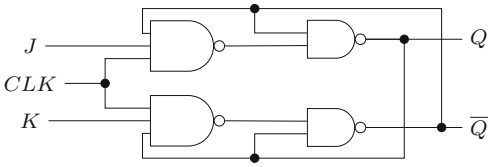


Fig. 5 J-K flip-flop block diagram

6.2.1 Fuzzy NAND gate

Fuzzy operations are simple to realize physically, except for more complex gates like eXclusive-OR. Variations of fuzzy NAND gates are implemented in [19], however they are ignored in this study because of their complexity. The NAND gate depicted in Fig. 5 is a fuzzy logic gate simply defined by min − max norms as follows:

$$\text{NAND}_1(x, y) = 1 - \min(x, y) = \neg(x \sqcap y)$$

This gate can also be defined using the Łukasiewicz t-norm as follows:

$$\text{NAND}_2(x, y) = 1 - \max(0, \min(x + y, 1)) = \wr\neg x + \neg y\int$$

6.2.2 Properties of fuzzy J-K flip-flop

Obviously, inputs and outputs of fuzzy NAND gate are decimals from [0, 1]. Suppose numbers smaller or equal to 0.25 as low and numbers larger or equal to 0.75 as high and the numbers in-between these two bounds as invalid values. According to Fig. 5 the following properties holds for each fuzzy J-K flip-flop.

Property 4 *If J is high and K is low at the clock edge, then Q output is forced high and stays high while $\overline{Q}$ is forced low and stays low for sure. Note that at the beginning initial values of Q and $\overline{Q}$ are random and even may be invalid values like* 0.5.

The following is FzCTL proposition of the above property:

$$\begin{aligned} P_1 =& J \geq 0.75 \sqcap K \leq 0.25 \\ & \rightarrow \text{AF}\left(\text{AG}(Q \geq 0.75 \sqcap \overline{Q} \leq 0.25)\right) \end{aligned}$$

Following is the FzPG for Fuzzy J-K flip-flop as depicted in Fig. 6,

$$\begin{aligned} G &= \left\{\{s_0\}, s_0, \{J, K, Q, \overline{Q}\}, \text{Init}, \text{Act}\right\} \\ \text{Init}(s_0) &= [1] \\ \text{Act}(s_0, s_0) &= ([1], J, K, Q_{next}, \overline{Q}_{new}), \\ Q_{next} &= \text{NAND}(\overline{Q}, \text{NAND}(\overline{Q}, J)), \\ \overline{Q}_{next} &= \text{NAND}(Q, \text{NAND}(Q, K)) \end{aligned}$$

If we rewrite NAND functions (as shown below), we can use them to construct the corresponding FzPG. Discrete saturation operator is used to quantize input values in order to have a finite number of states in the equivalent FzKripke.

$$\begin{aligned} &\text{NAND}_1(x, y) = \neg\left(\wr x\int_\varepsilon \sqcap \wr y\int_\varepsilon\right) \\ &\text{NAND}_2(x, y) = \wr\neg\wr x\int_\varepsilon + \neg\wr y\int_\varepsilon\int \quad \text{where } \varepsilon = 2^{-d}, d \geq 2 \end{aligned}$$

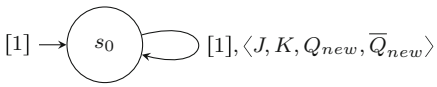


Fig. 6 FzPG of the J-K flip-flop shown in Fig. 5

Table 4 Unstable condition of Fuzzy J-K flip-flop using NAND_1

Step	J	K	Q	$\overline{Q}$
0	0.75	0.25	0.5	0.625
1	0.75	0.25	0.625	0.5
2	0.75	0.25	0.5	0.375
3	0.75	0.25	0.625	0.375
4	0.75	0.25	0.5	0.375
5	0.75	0.25	0.625	0.375
6	0.75	0.25	0.5	0.375
⋮	⋮	⋮	⋮	⋮

Using NAND_1 while constructing FzPG G there is a condition in which flip-flop works improperly (and that is when the initial state values for the Q and $\overline{Q}$ are invalid) thus the property is incorrect and proposition P_1 evaluates to 0; see Table 4 for traces and configuration of model. By substitution of NAND_1 with NAND_2, proposition P_1 (for all $\varepsilon \geq 0.25$) evaluates to 1, and the property always holds.

If another criterion is imposed to proposition P_1, then the property holds for both NAND_1 and NAND_2. The following proposition corrects improper behavior of the flip-flop.

$$
\begin{aligned}
P_1' = {} & J \geq 0.75 \sqcap K \leq 0.25 \\
& \sqcap (Q \geq 0.75 \sqcup Q \leq 0.25) \sqcap (\overline{Q} \geq 0.75 \sqcup \overline{Q} \leq 0.25) \\
& \rightarrow \text{AF}\left(\text{AG}(Q \geq 0.75 \sqcap \overline{Q} \leq 0.25)\right)
\end{aligned}
$$

7 Conclusion and future work

We defined *Quantized FzKripke* as a variant of multi-valued Kripke structure on interval $[0, 1]_\Delta$. We also defined FzCTL as the modal temporal logic to express temporal properties of FzKripke. A series of operators like quasi-comparison, bounded-add, bounded-subtract, and scalar-multiplication on discrete interval are defined in FzCTL that are rarely seen in other logics introduced prior to this. This gives FzCTL the ability to specify sophisticated behaviors of fuzzy systems easily.

Having defined a powerful fuzzy temporal logic, we paired it with FzPG as the modeling structure. This is a more convenient structure to model fuzzy systems due to its compactness and readability. Moreover, we introduced *Quantized FzPG* such that it is convertible to Quantized FzKripke. In order to model check an FzCTL formula φ over an FzPG a translation to an equivalent FzKripke with an exponential blow-up was inevitable leading to state-space explosion. By the means of OBDD vectors, we devised a symbolic method that can perform direct checking of φ on a Quantized FzPG avoiding the aforementioned exponential blow-up.

In order to demonstrate the applicability of our method, we formally investigated the correctness of a multi-valued D Flip-Flop as a case study, and showed a flaw in the circuitry that is not seen in design phase with computer simulations. This flaw yields to a dynamic

hazard making the whole flip-flop unstable under certain initial conditions. Afterwards, by investigating a potential design for Fuzzy J-K Flip-Flops, we demonstrated how the proposed method can be useful in design phase of such intuitive circuits for analyzing their complex behaviors.

Considering the expressibility of FzCTL, compactness and readability of FzPG, our method is affordable in terms of memory and computational complexity. We believe further investigation is needed to compare our proposed method with a relevant method introduced in [22] in terms of expressiveness, scalability, and applicability for real-world scenarios. We believe it is not the case that these methods are not equivalent but making this comparison is initially difficult because the modeling structure in [22], i.e., FzKripke structure, is by an exponential factor larger than FzPG and the model checking method in [22] follows a different theoretical base.

Furthermore, an improved implementation for EX operator is proposed in Appendix A, which is of lower memory and computational complexity. Not only we analysed the worst case complexity of all algorithms in this paper but we also implemented our method and made an intuitive assessment. Appendix A.2 reads about practical performance analysis of the proposed method through an intuitive assessment.

We had covered all preliminaries about abstraction and approximation in fuzzy temporal logic and models in a discrete setup [26]; and now we covered a symbolic model checker that deals with state-space explosion in the same setup. The ultimate goal of this series of research is to apply formal verification in field of fuzzy control systems. To this purpose, it is required to define some extensions to our modeling, specification, and verification approach. For future research our intention is to perceive concrete time in form of fuzzy sets and also to include natural language propositions in an augmented fuzzy extended temporal logic.

Acknowledgements Open access funding provided by Graz University of Technology. The authors would like to thank anonymous reviewers for their valuable comments and suggestions that improved the quality of this manuscript.

A An alternative solution for EX operator

In the primary design of EX operator proposed in this paper, not only U and W are used but also auxiliary variables of U' and W' participate in construction of vector R. This redundancy leads to diagrams of larger size that makes it difficult for models of large state-space and large attribute-set to fit into memory. Similar issue is in place with λ diagram in line 3 of Algorithm 4. Moreover, another vector called D is involved in lines 4 and 5 of the same algorithm which intensifies the problem; therefore, OBDDs in primary design of EX operator are of maximum hight:

$$L = 2h + d' = 2n + (2k + 1)d'$$

In this section we introduce another design for this algorithm in which OBDDs are of maximum height:

$$L' = h + d' = n + (k + 1)d'$$

If k is a large number and n is a small number L' is approximately half of L. This means the size of largest OBDD in primary approach is reduced to its squared root by using the alternative algorithm, in exchange the computational complexity may increase which is fully addressed later in this section. In order to redesign the algorithm we will use the following property repeatedly:

Property 5 *For conjunction and disjunction of two real values with a third value we have:*

$$\begin{aligned} a \sqcup b \leq D &\Rightarrow a \leq D \wedge b \leq D \\ a \sqcup b \geq D &\Rightarrow a \geq D \vee b \geq D \\ a \sqcap b \leq D &\Rightarrow a \leq D \vee b \leq D \\ a \sqcap b \geq D &\Rightarrow a \geq D \wedge b \geq D \end{aligned}$$

Even by substituting $\leq$ with $<$ and $\geq$ with $>$, these equations hold.

For each pairs of nodes s and t in an FzPG we define:

$$\begin{aligned} &\mathrm{Act}(s,t) = (A,B) \Leftrightarrow A_{st} = A \wedge B_{st} = B \\ &c_{st} \stackrel{\text{def}}{=\!=} U(s) \wedge U'(t) \wedge (W' = B_{st}) \\ &a_{st} \stackrel{\text{def}}{=\!=} \textsc{vIf}(c_{st}, A_{st}, \textsc{False}) \end{aligned}$$

According to above definitions relation R is obtainable as follows:

$$\mathrm{R} = \bigsqcup_{s,t \,\in\, S} a_{st}.$$

For the sake of brevity, we denote by Y the combination of variables U and W like:

$$Y = \langle U; W \rangle.$$

Now using above definition we can rewrite lines 2 to 6 of Algorithm 4 as follows:

1: ...
2: $\theta' := \theta[Y'/Y]$
3: nop
4: $\psi_1 := \exists Y' \cdot (\theta' \sqcap R \geq D)$
5: $\psi_2 := \forall Y' \cdot (\theta' \sqcap R \leq D)$
6: $\psi := \psi_1 \wedge \psi_2$
7: ...

According to properties of universal and existential quantifiers we have:

$$\neg\psi_2 = \exists Y' \cdot (\theta' \sqcap R > D).$$

By defining φ_1 and φ_2 like

$$\begin{aligned} \varphi_1 &\stackrel{\text{def}}{=\!=} \theta' > D \\ \varphi_2 &\stackrel{\text{def}}{=\!=} \theta' \geq D, \end{aligned}$$

we can rewrite ψ_1 as follows:

$$\psi_1 = \exists Y' \cdot (\theta' \geq D \wedge \mathrm{R} \geq D) = \exists Y' \cdot (\varphi_2 \wedge \bigsqcup_{s,t \in S} a_{st} \geq D) = \exists Y' \cdot (\varphi_2 \wedge \bigvee_{s,t \in S} a_{st} \geq D)$$

$$= \exists Y' \cdot \left(\bigvee_{s,t \in S} (\varphi_2 \wedge a_{st} \geq D) \right) = \bigvee_{s,t \in S} \exists Y' \cdot (\varphi_2 \wedge a_{st} \geq D)$$

similarly, we have:

$$\neg \psi_2 = \bigvee_{s,t \in S} \exists Y' \cdot (\varphi_2 \wedge a_{st} > D);$$

on the other hand:

$$\begin{aligned}(a_{st} \geq D) = \textsc{vIf}(c_{st}, A_{st}, \textsc{False}) \geq D &= \mathrm{if}(c_{st}, A_{st} \geq D, \textsc{False} \geq D) \\ &= \mathrm{if}(c_{st}, A_{st} \geq D, D = \textsc{False})\end{aligned}$$

now according to following definition

$$\overline{D} = (D = \textsc{False}) = \neg z_d \wedge \cdots \wedge \neg z_0$$

we have:

$$(a_{st} \geq D) = \mathrm{if}(c_{st}, A_{st} \geq D, \overline{D}) = c_{st} \wedge (A_{st} \geq D) \vee \neg c_{st} \wedge \overline{D}$$

which implies:

$$\overline{D} \Rightarrow A_{st} \geq D.$$

Meanwhile, following conditional statement is a tautology:

$$(q \Rightarrow p) \Rightarrow \big((r \wedge p) \vee \neg(r \wedge q) = (r \wedge p) \vee q\big)$$

Now we may conclude:

$$(a_{st} \geq D) = \overline{D} \vee c_{st} \wedge (A_{st} \geq D)$$

similarly we may write:

$$(a_{st} > D) = \mathrm{if}(c_{st}, A_{st} > D, \textsc{False} > D) = \mathrm{if}(c_{st}, A_{st} > D, 0) = c_{st} \wedge (A_{st} > D).$$

By defining P_{st} as the inner part of ψ_1 like

$$P_{st} \stackrel{\text{def}}{=\!=} \exists Y' \cdot (\varphi_2 \wedge (a_{st} \geq D))$$

we have:

$$\begin{aligned}P_{st} &= \exists Y' \cdot (\varphi_2 \wedge (\overline{D} \vee c_{st} \wedge (A_{st} \geq D))) = \exists Y' \cdot ((\varphi_2 \wedge \overline{D}) \vee (\varphi_2 \wedge c_{st} \wedge (A_{st} \geq D))) \\ &= \exists Y' \cdot (\varphi_2 \wedge \overline{D}) \vee \exists Y' \cdot (\varphi_2 \wedge c_{st} \wedge (A_{st} \geq D)).\end{aligned}$$

The following conditional statement is easily verified:

$$(\overline{D} \Rightarrow \theta' \geq D) \Rightarrow (\varphi_2 \wedge \overline{D} = \overline{D})$$

As can be seen $(A_{st} \geq D)$ is independent from Y' therefore we can unnest it from existential quantification with following definitions:

$$E_{st} \stackrel{\text{def}}{=\!=} \exists Y' \cdot (\varphi_2 \wedge c_{st})$$
$$\beta_{st} \stackrel{\text{def}}{=\!=} (A_{st} \geq D) \wedge E_{st}$$

now we have:

$$P_{st} = \overline{D} \vee \beta_{st}$$

and with replacing c_{st} we have:

$$E_{st} = \exists Y' \cdot (\varphi_2 \wedge U(s) \wedge U'(t) \wedge (W' = B_{st}))$$

now we can unnest $U(s)$ from $\exists Y'$ as it is independent from Y' like:

$$\begin{aligned} E_{st} &= U(s) \wedge \exists Y' \cdot (\varphi_2 \wedge U'(t) \wedge (W' = B_{st})) \\ &= U(s) \wedge \exists W' \cdot \exists U' \cdot (\varphi_2 \wedge U'(t) \wedge (W' = B_{st})) \end{aligned}$$

as can be seen $(W' = B_{st})$ is independent from U' thus can be unnested from $\exists U'$. We can rewrite E_{st} using following definition as

$$\eta_{st} = \exists U' \cdot (\varphi_2 \wedge U'(t))$$
$$E_{st} = U(s) \wedge \exists W' \cdot ((W' = B_{st}) \wedge \eta_{st})$$

Now regarding the famous one-point rule we may replace W' with B_{st} assuming:

$$B_{st} = \langle b_1; \ldots; b_k \rangle$$
$$W' = \langle W'_1; \ldots; W'_k \rangle$$

regarding the following definition

$$\hat{\eta}_{st} \stackrel{\text{def}}{=\!=} \eta_{st}[B_{st}/W'] = [b_1/W'_1; \ldots; b_k/W'_k]$$

we have:

$$\beta_{st} = (A_{st} \geq D) \wedge U(s) \wedge \hat{\eta}_{st}$$

Similarly above arguments can be repeated for $\neg\psi_2$ with following definitions:

$$\alpha_{st} \stackrel{\text{def}}{=\!=} \exists Y'.(\varphi_1 \wedge (a_{st} > D))$$
$$\gamma_{st} \stackrel{\text{def}}{=\!=} \exists U'.(\varphi_1 \wedge U'(t))$$
$$\hat{\gamma}_{st} \stackrel{\text{def}}{=\!=} \gamma_{st}[B_{st}/W']$$

Now by substituting $a_{st} > D$ and c_{st} we have:

$$\begin{aligned} \alpha_{st} &= \exists Y' \cdot (\varphi_1 \wedge c_{st} \wedge (A_{st} > D)) = (A_{st} > D) \wedge \exists Y' \cdot (\varphi_1 \wedge c_{st}) \\ &= (A_{st} > D) \wedge U(s) \wedge \hat{\gamma}_{st} \end{aligned}$$

Eventually, we can rewrite ψ like:

$$\psi = \psi_1 \wedge \psi_2 = \bigvee_{s,t \in S} (\overline{D} \vee \beta_{st}) \wedge \neg \bigvee_{s,t \in S} \alpha_{st} = \left(\overline{D} \vee \bigvee_{s,t \in S} \beta_{st} \right) \wedge \neg \bigvee_{s,t \in S} \alpha_{st}$$

and according to Algorithm 4 we have:

$$\forall i \in 0 \ldots d \cdot \rho[i] = \exists D \cdot (z_i \wedge \psi)$$

consequently

$$\rho[i] = \exists D \cdot \left(z_i \wedge \neg \bigvee_{s,t \in S} \alpha_{st} \wedge \left(\overline{D} \vee \bigvee_{s,t \in S} \beta_{st} \right) \right)$$

$$= \exists D \cdot \left(z_i \wedge \overline{D} \wedge \neg \bigvee_{s,t \in S} \alpha_{st} \vee z_i \wedge \neg \bigvee_{s,t \in S} \alpha_{st} \wedge \bigvee_{s,t \in S} \beta_{st} \right)$$

according to definition $z_i \wedge \overline{D} = 0$ therefore above equation can be simplified as:

$$\rho[i] = \exists D \cdot \left(z_i \wedge \neg \bigvee_{s,t \in S} \alpha_{st} \wedge \bigvee_{s,t \in S} \beta_{st} \right)$$

and by the following definition

$$\hat{\psi} = \neg \bigvee_{s,t \in S} \alpha_{st} \wedge \bigvee_{s,t \in S} \beta_{st}$$

we have:

$$\rho[i] = \exists D \cdot (z_i \wedge \hat{\psi}).$$

In order to compute $\hat{\psi}$ it is required to calculate α_{st} and β_{st} for edges of the Fuzzy Program Graph. Finally, we show how to compute ρ using $\hat{\psi}$ instead of using ψ. With the above premises we can rewrite Algorithm 4 as below:

Algorithm 8 $\rho(U, W) = \text{EX}\big(\theta(U, W)\big)$

1: **procedure** $\rho(U, W)$
2: $\theta' := \theta[U'/U, W'/W]$
3: $\theta_1 := \text{vGRT}(\theta', D)$
4: $\theta_2 := \text{vGEQ}(\theta', D)$
5: $\alpha := 0$
6: $\beta := 0$
7: **for** each transition edge $Act(s, t) = (A, \langle B_1, \ldots, B_k \rangle)$ **do**
8: $\gamma := \exists U' \cdot (U'(t) \wedge \theta_1)$
9: $\eta := \exists U' \cdot (U'(t) \wedge \theta_2)$
10: $\hat{\gamma} := \gamma[B_1/W'_1, \ldots, B_k/W'_k]$
11: $\hat{\eta} := \eta[B_1/W'_1, \ldots, B_k/W'_k]$
12: $\alpha := \alpha \vee (\hat{\gamma} \wedge \text{vGRT}(A, D) \wedge U(s))$
13: $\beta := \beta \vee (\hat{\eta} \wedge \text{vGRT}(A, D) \wedge U(s))$
14: **end for**
15: $\psi := \beta \wedge !\alpha$
16: **for** $i := 0 \ldots d$ **do**
17: $\rho[i] := \exists D \cdot (D[i] \wedge \psi)$
18: **end for**
19: **end procedure**

In line 7 we assume Act is a total function; however, in case of a partial function or relation we may replace the following line:

for each transition edge $((s,t) = (A, \langle B_1, \ldots, B_k \rangle)) \in \text{Act}$ **do**

$\vdots$

end for

A.1 Computational complexity

In above algorithm, line 2 is of time complexity of $O(d'2^h)$, lines 3 and 4are of time complexity of $O(d'2^{2(h+d')})$. The for loop in line 7 iterates for r times, lines 8 and 9 are restrictions on OBDD diagram θ_1 and θ_2 and take computational time proportional to the size of these diagrams, both of which are of memory complexity of $O(2^{h+d'})$. In lines 10 and 11we have simultaneous replacement of diagrams with variables; a recursive algorithm to this purpose is proposed in [16] called "bdd_compose". The structure of this algorithm for replacing N variables according to recursive relation is as follows:

$$\varphi[\alpha_i/y_i, \ldots, \alpha_N/y_N] = \neg\alpha_i \wedge \varphi[0/y_i][\alpha_{i+1}/y_{i+1}, \ldots, \alpha_N/y_N] \vee \alpha_i \wedge \varphi[1/y_i][\alpha_{i+1}/y_{i+1}, \ldots, \alpha_N/y_N]$$

However, we assume variables y_i in diagram φ are prioritized by their index, from most significant y_1 to least significant y_N, and they also have higher priorities over other participant variables in φ. We also assume variables y_i do not participate in diagrams α_i. Let the reminder of participant variables in diagram φ be z_1 to z_D and also let the participant variables in each α_i be a subset of variables x_1 to x_H then the total replacement time equates to $T(N)$ which is expressible by the following recursive relation:

$$T(N) = O(2^{2(D+H)}) + \begin{cases} 2T(N-1), & \text{N}> 0; \\ 0, & \text{N} = 0. \end{cases}$$

Since y_i has higher priority than y_{i+1}, computing $\varphi[0/y_i]$ and $\varphi[1/y_i]$ is of time complexity of $O(1)$. However, diagrams α_i are of memory complexity of $O(2^H)$ (which also is of complexity of $O(2^{D+H})$); operands of $\vee$ are of memory complexity of $O(2^{D+H})$ (due to replacement of y_i with x_j) which makes this operation of computational complexity of $O(2^{2(D+H)})$; the computational complexity of $\wedge$ is of $O(2^{2(D+H)})$ for similar reason. Accordingly the result of above relation is $O(2^N.2^{2(D+H)}) = O(2^{N+2(D+H)})$. According to above preparations lines 10 and 11 are of $O(2^{(2k+1)d'})$ because each W'_i is composed of d' variables therefore there are a total of $N = kd'$ variables in W'. Apart from W', the variable list D (which is composed of d' variables) is participating in diagrams η and γ. Meanwhile the variable list W (with kd' variables) is participating in each B_i. Total complexity of these lines are of $O(2^{kd'} + 2^{(d'+kd')})$ which equates to $O(2^{(2k+1)d'})$.

In line 12, A is of height kd' and D is of height d' therefore vGrt is of $O(d'2^{2d'(k+1)})$.$\hat{\gamma}$ and vGrt(A, D) are of height $(k+1)d'$, consequently their composition can be done in time $O(d'2^{2d'(k+1)})$. Computing $U(s)$ takes a little time with a simple diagram of memory complexity of $O(n)$ therefore its composition with $\hat{\gamma} \wedge$ vGrt(A, D) is of computational complexity of $O(n2^{(k+1)d'})$. Eventually total complexity of line 12 is as follows:

$$O\left(d'2^{2d'(k+1)} + n2^{(k+1)d'} + 2^{2(h+d')}\right) = O\left(2^{2d'(k+1)}(d' + 2^{2n})\right).$$

Likewise, line 13 is of the same computational complexity. Considering the operands of conjunction operator in line 14, this line is of complexity of $O(2^{2(h+d')})$ and ψ is of memory

Table 5 Performance evaluation of proposed method, assuming $n = 2, k = 8$

d	h	#EX	Time (s)	#Nodes	$\mathbb{P}(\varphi)$
2	26	5	0.06	24,275	0.25 ± 0.25
3	34	6	1.91	1,230, 107	0.375 ± 0.125
4	42	6	149.00	34,058,231	0.500 ± 0.0625
5	50	7	2426.00	35,561,429	0.46875 ± 0.03125

complexity of $O(2^{h+d'})$. Similar to the primary algorithm, ρ is obtainable in $O(d'2^{h+d'})$ by having ψ computed. Total complexity for this operator is as follows:

$$O\Big(d'2^h + d'2^{2(h+d')} + 2^{h+d'}\Big) + O\Big(r\Big(2^{(2k+1)d'} + 2^{2d'(k+1)}\big(d' + 2^{2n}\big)\Big)\Big)$$
$$+ O\Big(2^{2(h+d')} + d'2^{h+d'}\Big) = O\Big(r2^{2d'(k+2)}\big(d' + 2^{2n}\big) + d'2^{2(h+d')}\Big)$$

Usually d' is a small number comparing to $|S|$ therefore it is safe to assume $d' = O(2^n)$. In most cases, $r = O(|S|^2) = O(2^{2n})$ the overall time complexity of proposed algorithm for EX operator is of $O(2^{2(h+d'+n)})$. The AX is directly obtainable from EX operator therefore it is of the same computational complexity as EX. Time complexity of AU and EU operators are $O(2^{h+d'})$ times the computational complexity of EX operator. Since no preprocessing is required if using the latter implementation of EX operator the total time complexity of model checking process is as follows:

$$O(|\varphi|(2^{3h+3d'+2n}) + (d'2^{2(h+d')})) = O(|\varphi|.2^{3h+3d'+2n}).$$

The upper bound of time complexity of EX operator for the first method is $O(d'2^{4h})$ and for the second method is $O(2^{2(h+d'+n)})$ which is far less than the former. It is expected that the second method is of lower memory complexity due to lower height of diagrams.

A.2 Performance evaluation: an intuitive assessment

As the behavior of OBDD is unpredictable the assumptions over the complexity orders of proposed algorithms are not reliable. These orders are not tight and there are certain conditions and scenarios in which the first implementation of EX operator outperforms the second implementation. Alternatively, in this subsection we provide an intuitive assessment of our proposed method using the secondary definition of EX operator. We ran the experiments on an execution platform similar to the one that we described in Sect. 6.1.2.

Since there is no benchmark to evaluate our method we perform assessment using random data. We create a random single-source graph like $G = (S, s_0, X, \mathrm{Act})$ with following restrictions:

$$S = \{0, \ldots, N-1\}, \quad N = 2^n, \quad s_0 = 0, \quad X = \{x_1, \ldots, x_k\}$$
$$\mathrm{Act}(0,1) = \left(\prod_{r=1}^{k} (\gamma'_r \preceq x_r \sqcap \gamma''_r \succeq x_r), r =_1^k \langle x_r \rangle\right)$$
$$\mathrm{Act}(i,j) = (x_{p_{ij}} \sqcup x_{p'_{ij}} \sqcup \alpha_{ij}, r =_1^k \langle \lceil x_{q_{ijr}} \sqcup x_{q'_{ijr}} \sqcup \beta_{ijr} + \lambda_{ijr} \rfloor \rangle), \quad \forall i, j \in \{1, \ldots, N-1\}$$

All participant numbers in above relations are factors of Δ as follows:

$$\begin{aligned}&\forall r \in \{1, \ldots, k\}.\gamma'_r, \gamma''_r \in [0, 1]\\ &\quad \wedge \forall i, j \in \{1, \ldots, N-1\}.\alpha_{ij} \in [0, 1] \wedge p_{ij}, p'_{ij} \in \{1, \ldots, k\}\\ &\quad \wedge (\forall r \in \{1, \ldots, k\}.\beta_{ijr} \in [0, 1] \wedge \lambda_{ijr} \in [-1, 1] \wedge q_{ijr}, q'_{ijr} \in \{1, \ldots, k\})\end{aligned}$$

In order to evaluate the performance of our method, an FzCTL formula φ is required to be verified over above-defined random graph. φ is defined as follows:

$$\varphi = \mathrm{AX}\left(\mathrm{EF}\left(\prod_{r=1}^{k}(\gamma_r \succeq x_r)\right)\right)$$

where γ_r are random numbers belong to interval $[0, 1]_\Delta$. By manipulating parameters n and k and considering different precisions of d, several models are obtainable, all of which different from one another in terms of execution time and memory used. In our experiments we used permutations of $n \in \{1, \ldots, 5\}$, $k \in \{4, 8, 12, 16, 20\}$, and $d \in \{2, \ldots, 6\}$; some of which failed due to lack of allocated memory. If $d' = d + 1$ and $h = n + kd'$ then all experiments with $h \leq 50$ succeeded while consuming about 2GBytes of allocated memory. Table 5 demonstrates execution time, number of nodes (#Nodes), number of calls to EX operator (#EX) and obtained truth degree of φ for an experiment setup of $n = 2$ and $k = 8$. Total execution time in proportion to number of calls to EX operator depicts the average elapsed time for EX operator. As can be observed, incrementing d will result in more accuracy and better precision yet the execution time grows exponentially. In a nutshell, considering all these parameters, we automatically verified φ on models whose size are of $O(2^{50})$ on a fairly weak laptop with an acceptable accuracy and precision.

References

1. Avrachenkov, K., Sanchez, E.: Fuzzy markov chains and decision-making. FO & DM **1**(2), 143–159 (2002). https://doi.org/10.1023/A:1015729400380.
2. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press (2008). https://doi.org/10.1093/comjnl/bxp025. http://mitpress.mit.edu/books/principles-model-checking
3. Baresi, L., Pasquale, L., Spoletini, P.: Fuzzy goals for requirements-driven adaptation. In: 2010 18th IEEE International Requirements Engineering Conference, pp. 125–134. IEEE (2010). https://doi.org/10.1109/RE.2010.25. http://staff.lero.ie/lpasqua/files/2012/02/RE2010.pdf, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5636887
4. Bianco, A., de Alfaro, L.: Model Checking of Probabilistic and Nondeterministic Systems, pp. 499–513. Springer, Berlin (1995). https://doi.org/10.1007/3-540-60692-0_70
5. Bruns, G., Godefroid, P.: Model checking with multi-valued logics. In: Proceedings of Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12–16, 2004, pp. 281–293 (2004). https://doi.org/10.1007/978-3-540-27836-826
6. Chechik, M., Devereux, B., Easterbrook, S., Gurfinkel, A.: Multi-valued symbolic model-checking. ACM Trans. Softw. Eng. Methodol: TOSEM **12**(4), 371–408 (2003)
7. Chechik, M., Gurfinkel, A., Devereux, B., Lai, A., Easterbrook, S.: Data structures for symbolic multi-valued model-checking. Form. Methods Syst. Des. **29**(3), 295–344 (2006). https://doi.org/10.1007/s10703-006-0016-z
8. Choi, B., Shukla, K.: Multi-valued logic circuit design and implementation. Int. J. Electron. Electr. Eng. **3**(4), 256–262 (2015)
9. Frigeri, A., Pasquale, L., Spoletini, P.: Fuzzy time in linear temporal logic. ACM Trans. Comput. Log. **15**(4), 30:1–30:22 (2014). https://doi.org/10.1145/2629606
10. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Form. Asp. Comput. **6**(5), 512–535 (1994). https://doi.org/10.1007/BF01211866

11. Henzinger, T.A.: The theory of hybrid automata. In: Proceedings of 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27–30, 1996, pp. 278–292. IEEE Computer Society (1996). https://doi.org/10.1109/LICS.1996.561342
12. Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? J. Comput. Syst. Sci. **57**(1), 94–124 (1998). https://doi.org/10.1006/jcss.1998.1581. http://www.sciencedirect.com/science/article/pii/S0022000098915811
13. Hirota, K., Ozawa, K.: The concept of fuzzy flip-flop. IEEE Trans. Syst. Man Cybern. **19**(5), 980–997 (1989). https://doi.org/10.1109/21.44013. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=44013'escapeXml='false'/>
14. Hirota, K., Pedrycz, W.: Design of fuzzy systems with fuzzy flip-flops. IEEE Trans. Syst. Man Cybern. **25**(1), 169–176 (1995). https://doi.org/10.1109/21.362956
15. Liang, W., Bing-wen, W., Yi-Ping, G.: Cell mapping description for digital control system with quantization effect. Technical report (2007). http://arxiv.org/abs/0712.2501
16. Lind-Nielsen, J.: BuDDy—A Binary Decision Diagram Package. IT-TR. Department of Information Technology, Technical University of Denmark, Lyngby (1996)
17. Moon, S., Lee, K.H., Lee, D.: Fuzzy branching temporal logic. IEEE Trans. Syst. Man Cybern. Part B **34**(2), 1045–1055 (2004)
18. Norman, G., Parker, D., Sproston, J.: Model checking for probabilistic timed automata. Form. Methods Syst. Des. **43**(2), 164–190 (2013)
19. Ozawa, K., Hirota, K., Koczy, L.: Fuzzy flip-flop. In: Patyra, M.J., Mlynek, D.M. (eds.) Fuzzy Logic. Implementation and Applications, pp. 197–236. Wiley, Chichester (1996)
20. Ozawa, K., Hirota, K., Koczy, L.T., Pedrycz, W., Ikoma, N.: Summary of fuzzy flip-flop. In: Proceedings of 1995 IEEE International Conference on Fuzzy Systems, vol. 3 (1995). https://doi.org/10.1109/FUZZY.1995.409897
21. Palshikar, G.K.: Representing fuzzy temporal knowledge. In: International Conference on Knowledge-Based Systems (KBCS-2000), pp. 252–263. Mumbai, India (2000)
22. Pan, H., Li, Y., Cao, Y., Ma, Z.: Model checking fuzzy computation tree logic. Fuzzy Sets Syst. **262**, 60–77 (2015)
23. Pedrycz, W.: Fuzzy Control and Fuzzy Systems (*2nd*, extended edn. Research Studies Press Ltd., Taunton (1993)
24. Pnueli, A.: Current trends in concurrency. Overviews and tutorials. In: Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends, pp. 510–584. Springer, New York (1986). http://dl.acm.org/citation.cfm?id=19518.19527
25. Sladoje, N.: On analysis of discrete spatial fuzzy sets in 2 and 3 dimensions. Ph.D. thesis, Swedish University of Agricultural Sciences Uppsala (2005). http://pub.epsilon.slu.se/963/1/ThesisNSladoje.pdf
26. Sotudeh, G., Movaghar, A.: Abstraction and approximation in fuzzy temporal logics and models. Form. Asp. Comput. **27**, 1–26 (2014). https://doi.org/10.1007/s00165-014-0318-7
27. Whittle, J., Sawyer, P., Bencomo, N., Cheng, B.H., Bruel, J.M.: RELAX: incorporating uncertainty into the specification of self-adaptive systems. In: 2009 17th IEEE International Requirements Engineering Conference, pp. 79–88. IEEE, Atalanta, Georgia (2009). https://doi.org/10.1109/RE.2009.36
28. Wierman, M.: An Introduction to the Mathematics of Uncertainty, p. 133. Creighton University (2010). http://typo3.creighton.edu/fileadmin/user/CCAS/programs/fuzzy_math/docs/MOU.pdf
29. Zadeh, L.A.: Probability Theory and Fuzzy Logic. https://pdfs.semanticscholar.org/38a2/90e1d109427869ca95904457e45c2265d117.pdf. Accessed 23 Jan 2017