Real-Time Systems Modeling and Analysis*

Lakhan Shiva Kamireddy[0000−0001−6007−5408]

University of Colorado, Boulder CO 80302, USA
{lakhan.kamireddy}@colorado.edu

Abstract. This paper is a survey of extensions to finite automata theory to model real-time systems as well as systems exhibiting mixed discrete-continuous behavior. Real-time systems maintain a continuous and timely interaction with the environment, often adhering to some timing constraints. Therefore, the finite automata theory is extended to measure real-time values and accept or reject runs on a class of automata known as timed-automata, upon satisfying some timed properties. The automata modeling the mixed discrete-continuous behavior of hybrid systems has its continuous-time dynamics described using ordinary differential equations for the state space and discrete-time dynamics describing the control decisions. Based on these dynamical system models, we likewise extend the finite automata theory to describe the behavior of hybrid systems using Hybrid Automata. We further study some applications of this class of systems, sometimes referred to as Cyber-physical systems and perform a case-study on Peterson's Mutual Exclusion protocol using Uppaal.

Keywords: Real-time systems · Timed Automata · Hybrid Automata · Cyber-Physical Systems · Uppaal.

1 Introduction

1.1 Real-time systems and automata based modeling

Real-time systems maintain timely interaction with the environment. The timing constraints are crucial to such systems and it may lead to dramatic consequences when these constraints are not met. We therefore would like to model such systems using automata theory and model check the timed properties of real-time systems. However, we don't have the necessary resources to model time in finite automata. In 1994, Alur and Dill published the results of their study as the theory of timed automata [1], thereby solving this problem. Timed automata theory is an extension to finite automata theory that describes a way of measuring real-time event occurrences in such systems. It equips us with the required resources in verifying these real-time systems and obtaining provably correct guarantees of timed properties in them. We will study various properties of timed automata and prove some of them in section II. We will also look at a class of systems

* University of Colorado Boulder

known as Hybrid Systems that have mixed discrete-continuous behavior. We will look at the hybrid automata theory for modeling this behavior in section III. In section IV we will look at some of the applications of systems known as Cyber-Physical systems that have these hybrid properties. In section V we look at a case-study analysis of the peterson's mutual exclusion protocol in a tool for modeling and verifying real-time systems, and discuss open problems and conclusions in section VI.

Real-time systems Real-time systems are encountered in many instances, and we interact with them more often than we realize. Some examples of real-time systems are event response systems like airbag systems, closed-loop control like cruise control system in a car, aircraft control systems, cardiac pacemakers. Before looking at automata for modeling real-time systems, let us first take a look at a class of words that have an embedded time component within them.

Timed words An alphabet Σ is defined over a finite set of letters. A timed word is a tuple (w, t) where w is a word over the alphabet Σ, w=$a_1a_2a_3...a_n$ and t is a time sequence, t=$t_1t_2t_3...t_n$ where $t_1 \leq t_2 \leq t_3 \leq ...t_n$ and $t_i \in R_{\geq 0}$, the set of non-negative real numbers. A timed language is a set of words in L such that $L \in T\Sigma^*$ is a property over timed words [2].

Automaton modeling timed properties Fig. 1. shows an automaton modeling a lamp. It also describes the timed properties of the lamp. The lamp starts in off state. When the switch is pressed in off state, it takes the transition to low state while resetting a clock, y to zero. From low state if the switch is pressed again before 5 seconds have elapsed, it transitions to bright state. From low state if the switch is pressed after 5 seconds have elapsed, it takes a transition back to the off state and accepts. From bright state, if the switch is pressed, it transitions to off state and accepts.

1.2 Timed Automata

A timed automaton is composed of a finite automaton and a finite set of real-valued clocks. All the clocks values increase at the same rate. Guards can be placed on the transitions of the automaton using which we can enable or disable that transition, thereby constraining the behavior of the automaton and describing timed properties through the language of the automaton. Guards are comparisons of clock values with constants that are non-negative rational numbers and can either evaluate to true or false, g: $x \leq c \mid x \geq c \mid \neg g \mid g \wedge g$ where $x \in Clocks$, $c \in Q_{\geq 0}$. The clocks can be reset.

Formal defn. A timed automaton M is defined as a tuple, M = (Q, Σ, C, Inv, δ, q_0, F). Q is the finite set of states. Σ is a finite set of actions. C is a finite set of clocks. Inv associates each location with an invariant. δ is a set of transitions.

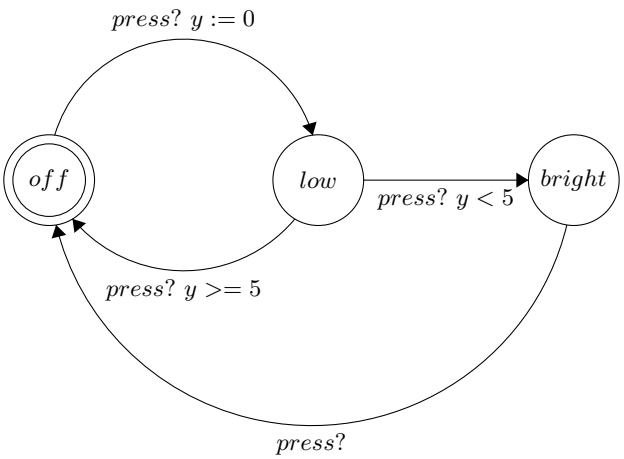

Fig. 1. Automaton modeling timed properties of a lamp

(q,a,g,r,q') is a transition from q to q', executing an action a, satisfying a guard g and clock resets in r. There are two types of transitions namely a location switch and a time switch. By elapsing time, and satisfying the location invariant, the automaton can stay in the same state thus making a time switch. By satisfying the guard and taking a transition to another location, the automaton makes a location switch.

2 Timed Automata properties

2.1 Non-deterministic timed automata

There can be non-determinism concerning location and non-determinism concerning time. ϵ transitions may also introduce non-determinism w.r.t location. Fig. 2. shows an example of location non-determinism. Upon reading a b from the state s_1 we cannot determine deterministically if the automaton is going to stay in the state s_1 or if it is going to transition to the state s_2. Fig. 3. shows an example of time non-determinism. Upon reading an a from the state s_1 we cannot determine deterministically the time of occurrence of the event (reading an a) as long as it satisfies the guard condition t ≤ 4 and location invariant t $\leq$ 5.

2.2 Deterministic timed automata

A timed automaton is deterministic if a) It has only one initial location, b) It doesn't have ϵ transitions, c) Event determinism: Two edges with same source and same label have disjoint guards ($g_1 \cap g_2 = \emptyset$), d) Time determinism: For every transition, the intersection of g with I_q is at most a singleton.

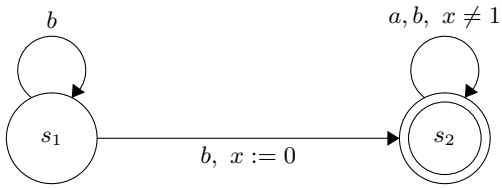

Fig. 2. Non-deterministic Timed Automaton (location non-determinism)

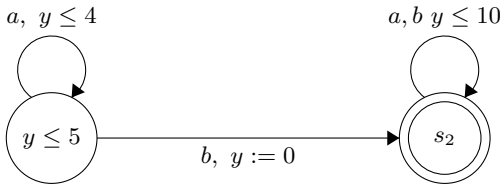

Fig. 3. Non-deterministic Timed Automaton (time non-determinism)

2.3 Expressiveness of ϵ transitions

ϵ transitions add to the expressiveness of a timed automaton, i.e. the language recognized by a timed automaton with ϵ transitions may not be recognized by a timed automaton without the ϵ transitions. Take the automaton in Fig. 4. as an example. This automaton accepts timed words over a such that every occurrence time is an integer and no two a events occur at the same time. This language cannot be accepted by a timed automaton if ϵ switches are not allowed. If the largest constant in such timed automaton is c, then a timed automaton without ϵ transitions cannot distinguish between the words (a, c+1) and (a, c+1.1). In this case, it can't distinguish between (a, 2) and (a, 2.1).

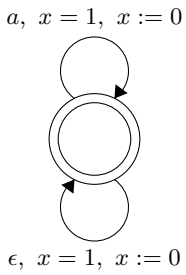

Fig. 4. Expressiveness of ϵ transitions

2.4 Timed regular languages

A timed language is called timed regular if it can be accepted by a timed automaton.

2.5 Closure properties

Theorem 1. *Timed regular languages are closed under the following operations.*

Let us consider two timed regular languages, L_1, L_2.

1. *Union*: Union of two timed regular languages, $L_1 \cup L_2$ is timed regular.
2. *Intersection*: Intersection of them, $L_1 \cap L_2$ is timed regular.
3. *Projection*: Projection of a timed regular language is timed regular.
4. *Untime*: If L is timed regular, then untime(L) is ω-regular [3].

Closure under Union and Intersection are established by constructing product of the timed automata. Projection can be proved by labeling transitions with ϵ. The proof for untime(L) being ω-regular is established by region construction as shown in [1].

2.6 Closure under complementation

Theorem 2. *Timed regular languages are not closed under complementation [3].*

Proof. Let Σ = {a, b}. L is a timed language consisting of timed words w, containing an a event at some time t such that no event occurs at time t + *1*. L is accepted by the timed automaton in Fig. 5. We will show that $\overline{L}$ is not timed regular. *untime(L)* accepts *(a+b)*a(a+b)**.

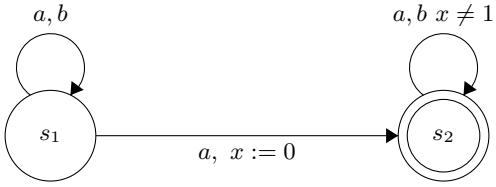

Fig. 5. Timed Automaton for disproving closure on complementation

Consider timed regular language L', consisting of timed words w, such that untimed word of w is in a^*b^*. All a events happen before time 1 and no two a events happen at same time. Observe the word $a^m b^n \in$ untime($\overline{L} \cap L'$) iff m $\geq$ n. Timed regular languages are closed under intersection, *untime(L)* of the timed regular language L is regular, but since we also know that $a^m b^n$ m $\geq$ n is not regular, we proved that $\overline{L}$ is not timed regular.

2.7 Emptiness

Given a timed automaton over finite words, is the language accepted by it empty? This problem is known as the **emptiness** problem for timed automata. It can be analyzed by checking if there is a run of the automaton from an initial state to a final state.

Proof. We have a problem in performing this check since the number of configurations is uncountably infinite due to real-valued clock times. The solution to this problem was proposed in [1] by constructing a finite region graph using region equivalence technique. The caveat here is that although there are infinite configurations, some clock valuations are equivalent and hence a timed automaton cannot distinguish between them. By grouping such configurations using region equivalence technique, we construct a finite region graph. The problem thus reduces to a reachability problem over this finite region graph, which is decidable. Hence it is proved that emptiness problem for timed automata is decidable.

2.8 Universality

Given a timed automaton over finite words, is the language accepted by it the set consisting of every timed word? This problem is known as the **universality** problem for timed automata. This problem can be proved undecidable by reducing the halting problem over two-counter machines, which is known to be undecidable to the universality problem [3].

2.9 Language inclusion

Given two timed automata A and B over finite words, checking if $\mathrm{L(A)} \subseteq \mathrm{L(B)}$ is known as the **language inclusion** problem. This problem can be proved undecidable by reducing the halting problem over two-counter machines, which is known to be undecidable to the language inclusion problem [3].

2.10 Decidability in special cases

Let us consider the universality problem on a given timed automaton having at most one clock. In this special case, this problem has been proved to be decidable by Abdulla et al. in [4]. Given two timed automata A and B, such that the timed automaton B only has one clock, and the only constant appearing in the clock constraints of B is 0, the language inclusion problem on A and B, formulated as checking if $\mathrm{L(A)} \subseteq \mathrm{L(B)}$ has been proved to be decidable by Ouaknine et. al. in [5].

2.11 Decidability of determinizability

In [6], E. Asarin has posed an open question concerning timed automata as follows. *Given a timed automaton A, is it possible to decide whether it is equivalent to a deterministic one?* This question remained open until O. Finkel has proved in [7] that this problem is undecidable.

3 Hybrid Systems

3.1 Background

Hybrid Systems are dynamical systems with interacting continuous-time dynamics and discrete-time dynamics. The evolution of the state of a continuous time system is described by an **ordinary differential equation** (ODE), $\dot{x} = Ax$, whereas the evolution of the state of a discrete-time system is described by a **difference equation**, $x_{k+1} = \mathrm{A}x_k$. The continuous time dynamics are referred to as flows and the discrete-time dynamics are referred to as jumps. The flows cause the system's state to make a smooth continuous transition and jumps cause the system to transition to a different set of flow equations by making a discrete jump to the new continuous-time dynamics. The hybrid behavior arrives in various contexts, a) Continuous systems with a phased operation, like the bouncing ball, biological cell growth to name a few, b) Continuous systems controlled by discrete logic, like control modes for complex systems, c) Coordinating processes, like multi-agent systems. When formally modeling the hybrid system behavior, the modeling paradigm we choose needs to have both continuous and discrete parts to it [8].

3.2 Bouncing ball

Let us consider an example of a ball of mass m freely falling from height $x \geq 0$ that bounces after hitting the ground. The analysis of this system has two parts to it.

Part-I Free Fall In free fall while x $\geq$ 0, the ball is under the influence of gravity. The equation of motion satisfies $\ddot{x}$ = -g, where x is the position of the ball and g is the gravitational constant. We reduce the order of the ODE by substituting $\dot{x}$ = v, where v is the velocity of the ball. We have $\dot{v}$ = -g. This describes the continuous time dynamics of the system.

Part-II Bouncing When the ball is at x = 0, and has a velocity downwards ($v < 0$), the ball bounces on the ground. There is a loss of velocity due to deformation and friction. The velocity discretely jumps to a different value as per the equation v := -cv, where $c < 1$. The velocity's magnitude is reduced and it's direction flips.

Hybrid Automaton The hybrid automaton for bouncing ball is illustrated in Fig. 6. x_1 represents the vertical position and x_2 represents the velocity of the ball [9]. In Fig. 7 we illustrate the bouncing ball simulation in MATLAB, where the yellow legend corresponds to position of the ball, and blue legend corresponds to velocity of the ball.

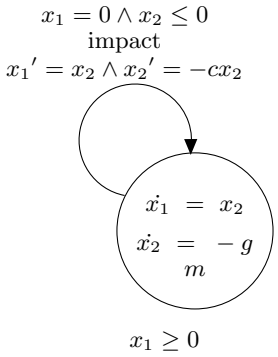

Fig. 6. Hybrid automaton

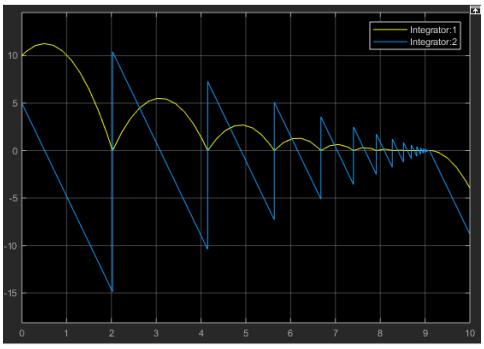

Fig. 7. Bouncing ball simulation

Formal defn. A hybrid automaton H is defined as a tuple, H = (M, Σ, M_0, X, δ, I, F, J). M is the finite set of control modes, also known as locations. Σ is a finite set of actions. Actions are given as sets of differential equations. M_0 is a finite initial set of control modes, $M_0 \subseteq M$. X is a finite set of real-valued variables. δ is a set of transitions. I is the mode-invariant function. F is the mode dependent flow function, characterizing flow at each mode. J is the jump function. The configuration of a hybrid automaton is (m, v) where $m \in M$ is a mode/location and v is a variable valuation.

Reachability Linear Hybrid Automata are a class of hybrid automata where its activities, invariants, and transition relations are defined by linear expressions. Given a linear hybrid automaton, is there a run reaching a particular state from the initial state? This problem is known as the **reachability** problem for linear hybrid automata. In [10], Alur et al. have reduced the halting problem of two-counter machines, which is known to be undecidable to the reachability problem for linear hybrid automata. Hence, it is proved to be undecidable.
Further, in [11], S.N. Krishna et al. have reduced the halting problem for two-counter machines to the reachability problem on recursive hybrid automata, thus showing that it is undecidable. However, bounded reachability is still decidable. There exist some incomplete algorithms for reachability. In [12], E. Abraham presents one such incomplete algorithm for linear hybrid automata based on fixed-point computation. Termination of this algorithm corresponds to finding the least fixed-point for the one-step forward reachability starting from the initial set.

4 Cyber-Physical Systems

4.1 Background

As defined in [13], a cyber-physical system consists of a collection of computing devices communicating with one another and interacting with the physical

world through sensors and actuators in a feedback loop. There are quite a few applications of such systems everywhere, whether it is smart buildings or medical devices or even automobiles. One of the distinguishing characteristics of cyber-physical systems is that they are reactive in nature, meaning these systems interact with the environment in an ongoing manner through inputs and outputs, like a program for a cruise controller in a car just to name one. They are concurrent in nature, meaning as opposed to sequential computation, there are multiple threads known as processes executing concurrently. They have mixed discrete-continuous behavior such as the hybrid systems. They often have real-time system characteristics and are in the most safety-critical areas where errors could lead to catastrophic eventualities. We will study some crucial applications of cyber-physical systems and look at automata based modeling and analysis of such systems.

4.2 Cardiac Pacemaker

Implantable cardiac pacemakers are life-saving devices that help patients with heart arrhythmia conditions. Ironically, there are bugs even in such safety-critical devices. To guarantee the correct operation of such devices, we need a model of the heart that captures the physiological conditions of the heart and respond to pacemaker outputs, using which we can verify the safety properties of the pacemaker. The pacemaker functions autonomously according to the device algorithm that is implemented onto it, interacts with its environment through sensors, actuators and other devices. It is, therefore a perfect example of a cyber-physical system as defined in the previous section. Timed automata is an appropriate formalism for such a heart model since most timing behaviors of heart can be captured by timed automata. In [14] Z. Jiang et. al. have proposed a real-time heart model based on timed automata and capture such crucial timing properties of the heart to then verify the cardiac pacemaker.

5 Uppaal case study

5.1 Background

Uppaal is a model checker for real-time systems. We can conveniently model a real-time system as a network of timed automata running concurrently. Uppaal is used to verify system properties specified in CTL over the timed automata system model [15]. The CTL formulae can be specified over paths, and are classified broadly as *safety*, *liveness* and *reachability* properties. Uppaal runs on a client-server architecture and is split into the backend model checking engine and frontend GUI communicating via TCP/IP protocol. It has a simulator where the user can run the system manually, and a verifier where the user may specify the properties and run the model checker on the system.

5.2 Peterson's mutual exclusion

The idea of mutual exclusion is that two processes which have critical sections cannot enter those sections at the same time. It is a technique for avoiding race conditions and keeping the result deterministic while running the processes concurrently. The Peterson's mutual exclusion algorithm is illustrated below.

Algorithm 1 Peterson's Mutex

1: **procedure** PROCESS 1	1: **procedure** PROCESS 2
2: req1 = 1;	2: req2 = 1;
3: turn = 2;	3: turn = 1;
4: while(turn!=1 && req2!=0);	4: while(turn!=2 && req1!=0);
5: *//critical section*	5: *//critical section*
6: job1();	6: job2();
7: req1 = 0;	7: req2 = 0;

Fig. 8. illustrates the results of Uppaal model checker on the correctly implemented mutex algorithm as above. First property species that there exists a run where critical section is reachable as $E <> (P1.CS)$. Second property captures the mutual exclusion property, where on all paths both P1 and P2 cannot be in the critical section at the same time as A[] not (P1.CS and P2.CS). We see that both the properties are satisfied in this case.

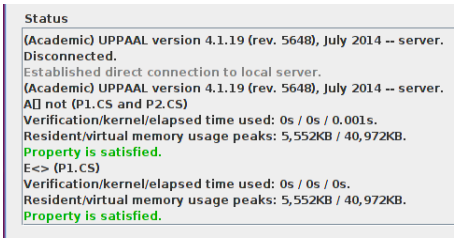

Fig. 8. Model checking mutex

Fig. 9. illustrates incorrectly implemented mutex, where two processes are seen to be in the *criticial section* at the same time in the simulator. Fig. 10 illustrates the results where Uppaal says that mutual exclusion property is not satisfied on the incorrectly implemented mutex. Mutual exclusion property has failed to satisfy the model.

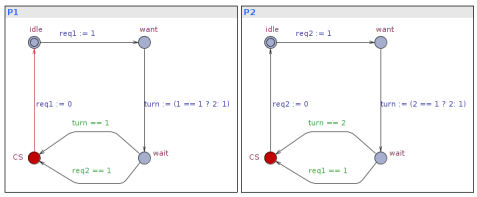

Fig. 9. Mutex protocol with bug

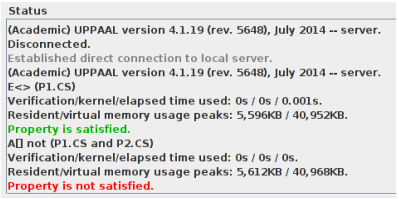

Fig. 10. Model checking mutex with bug

6 Conclusion

The motive of this work is to study extensions to finite automata theory used to model systems of the 21st century that are becoming increasingly complex to verify, due to the confluence of sensors, actuators, real-time constraints, and their concurrent and reactive nature. The automata theory based modeling paradigms discussed above are helping us design computer-based models of such systems that can be used to analyze their behavior mathematically and ultimately build safety-critical systems that are more dependable and secure although complex.

As we use model checking to verify the properties of such systems, we also see that some seemingly simple problems like reachability don't have a decidable algorithm that works for any general system. Researchers in this area are working towards devising algorithms to model check these systems. These algorithms although while being incomplete algorithms, terminate in most cases that are very relevant to the system analysis. Future work also lies ahead in researching methods to synthesize correct-by-construct designs for controllers and other components involved in such real-time hybrid systems. Some open problems in hybrid systems are concerning non-linear hybrid systems. Scalability is a challenge for most of the modeling tools available to us currently. Furthermore, high-dimensional systems pose a unique challenge to reachability analysis in these tools. Recent works such as [16], show promise in this direction. Schupp et al. [17] present an analysis of the current challenges in verification of hybrid systems.

References

1. Alur, Rajeev and Dill, David L., *A Theory of Timed Automata* 1994, Theor. Comput. Sci., vol. 126, pp. 183–235. Elsevier. https://doi.org/10.1016/0304-3975(94)90010-8.
2. Srivathsan, B., *Automata for Real-time Systems* 2013, CMI.
3. Alur, Rajeev and Madhusudan, P., *Decision Problems for Timed Automata: A Survey* 2004, Lecture Notes in Computer Science: Formal Methods for the Design of Real-Time Systems 3185, pp. 1–24. Springer.
4. Abdulla, P.A., Deneux, J., Ouaknine, J., Quaas, K., and Worrell, J., 2008. *Universality Analysis for One-Clock Timed Automata*, Inf. 89, 4 (2008), pp. 419–450. Fundam.

5. Ouaknine, J. and Worrell, J. *On the language inclusion problem for timed automata: closing a decidability gap*, Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science, 2004., pp. 54-63, Turku, Finland, https://doi.org/10.1109/LICS.2004.1319600.
6. Eugene Asarin. *Challenges in Timed Languages: From Applied Theory to Basic Theory*. Bulletin-European Association for Theoretical Computer Science; 1999, 2004, 83, pp. 106-120. $< hal - 00157685 >$.
7. Olivier Finkel. *Undecidable Problems About Timed Automata*. Eugene Asarin and Patricia Bouyer. FORMATS06, 2006, France. Springer, pp.187-199, 2006, Lecture Notes in Computer Science, Volume 4202. $< hal - 00121529v2 >$.
8. Lygeros, J., Tomlin, C., Sastry, S.: *Hybrid Systems: Modeling, Analysis and Control*. (2008).
9. Krishna, S.N. and Trivedi, A., *Hybrid Automata for formal modeling and verification of Cyber-Physical Systems* 2013, Journal of the Indian Institute of Science, vol. 93:3, pp. 419–440.
10. R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, *The algorithmic analysis of hybrid systems*. TCS, 138(1), 1995.
11. Krishna, S.N., Manasa, L. and Trivedi, A., *Improved Undecidability Results for Reachability Games on Recursive Timed Automata*. 2014, Electronic Proceedings in Theoretical Computer Science, vol. 161, pp. 245–259.
12. Abraham, Erika. *Modeling and Analysis of Hybrid Systems Lecture Notes*. RWTH Aachen University (2012).
13. Alur, Rajeev., *Principles of Cyber-Physical Systems*, 1st edn. MIT Press, Cambridge, MA (2015).
14. Jiang, Z., M. Pajic , A. Connolly , S. Dixit and R. Mangharam. *Real-time Heart Model for Implantable Cardiac Device Validation and Verification*. Presented at 22nd Euromicro Conference on Real-Time Systems (ECRTS10), 2010, Brussels, Belgium.
15. Behrmann, G., David, A. and Larsen, K. G. *A Tutorial on Uppaal*. Formal Methods for the Design of Real-Time Systems. Springer, pp.200-236, 2004.
16. Bak, Stanley, Tran, H-D., Johnson, Taylor T., *Numerical Verification of Affine Systems with up to a Billion Dimensions*. 2018. Accessed from https://arxiv.org/abs/1804.01583.
17. Schupp, S., Ábrahám, Erika, Chen, Xin, Ben Makhlouf, Ibtissem, Frehse, Goran, Sankaranarayanan, S., Kowalewski, S., *Current Challenges in Verification of Hybrid Systems*. 2015, In International Workshop on Design, Modeling, and Evaluation of Cyber Physical Systems, pp.8–24, Springer, Cham.