An automata theoretic framework for detecting schedulability attacks in cyber-physical systems

Abstract. Industrial Control Systems (ICS) used in manufacturing, power generators and other critical infrastructure monitoring and control are ripe targets for cyber attacks these days. Examples of such attacks are abundant such as attacks on Iranian nuclear enrichment plant with Stuxnet in 2009, on German steel plant in 2014, Ukrainian power system in 2015 and 2016. Usually in ICS, multiple control loops work concurrently and share various resources including the communication bus through which they interact with sensors and actuators. Real-time scheduling of concurrent control applications while competing for shared resources demands a delicate balance between performance and real-time constraints. A possible insider attack could be the replacement of a previously vetted control application or other components in the system, during a system update. In this paper, we propose an automated framework that addresses the effect of such replacement attacks from the perspective of loss of control performance. Given a set of control components, a control objective to be satisfied by the control ensemble, the question of schedulability and synthesis of a scheduler that can ensure the desired control performance has been recently studied in literature. In this paper, we extend this idea further to build an automata theoretic framework for assessment of replacement attacks on schedulability. We have built an end-to-end framework that takes in a set of control components, their variants (after replacement), a control objective to be guaranteed, and performs an automated schedulability assessment. We report some preliminary experiments of our framework on simple benchmarks.

1 Introduction

Most critical infrastructures such as power grid, water / sewage control, power generation plants, industrial automation etc., are cyber-physical systems (CPS). Cyber-Physical systems have two major interacting components – a component with physical dynamics governed by laws of physics – modeled with partial differential equations, another component – control, computing and networking which controls the physical dynamics. For very large and geographically distributed cyber physical systems, the control trajectory is designed to be stable and reliable with very complex distributed as well as centralized control. The control components today routinely involve a non-trivial amount of software, running on embedded control units on-board, to support and run any moderately sophisticated CPS infrastructure.

In recent history, cyber physical systems have been a popular victim of choice for so called cyber attacks, the effects of which have been moderate to as ravaging

as blackouts [1] or financial loss. This has prompted NIST [2] to promote cyber security frameworks for critical infrastructure [3] as one of the themes of immediate research attention. Given that no cyber security framework is full proof, significant research thrust is being invested in recent times to develop techniques that allow us to continually monitor the physical dynamics of these systems and look for any anomalies that could be indicative of cyber attack induced problems. Early detection of system dynamics changes would allow us to contain the damage by immediately islanding parts of the system which point to possible origin areas in the infrastructure.

Research in security of CPS is extremely crucial for developing novel technology for cyber attack detection, prevention, and countermeasures. Systematic studies on different sources of cyber attacks have revealed myriads of possibilities by which these cyber threats can propagate inside a CPS infrastructure. A person with local or remote access to the equipment of the physical plant, or access to the various interfaces such as programmable logic controllers (PLCs) or other Intelligent Electronic Device (IEDs) that are connected to the physical system for measurements and control can exploit a vulnerability in these devices to induce an attack on the system. One could also gain access to the control network, and create various kinds of man-in-the-middle attacks by either suppressing measurements or control actuation signals, replaying stale measurements or actuation signals, or even injecting maliciously planned false data. These kinds of attacks would then mislead the controllers, and wrong control actions could lead to disastrous industrial accidents. One could also hack into the controllers or the various other computing elements in the control center such as the process control servers by exploiting vulnerabilities in their design and attack the cyber physical system. In fact, in case of the Stuxnet worm [4], vulnerabilities in the Siemens SCADA system were exploited. While individual areas of cyber security research have received much attention in academia (e.g. cryptography; crypto-analysis; network security in the form of firewall and other perimeter security techniques, and intrusion detection, anti-virus software, and static analysis of software to detect vulnerabilities and zero-day attacks; hardware Trojan detection), much of these research focus on guarding IT systems in business and corporate information infrastructure.

Problem addressed in this paper: The focus of study in this paper is replacement attacks, wherein some or parts of a control component are compromised, or in other words, one or more control component is replaced with a modified one. In an industrial control environment with real time control components, replacements can lead to disaster, for example, slowing down a particular process in the industrial manufacturing can cascade a chain of failures in the whole assembly line. In this paper, we propose an idea that will demonstrate that such attacks for real-time SCADA systems can be guarded against by statically analyzing the legitimate control programs, and constructing an omega-regular language based timing signature, which can then be periodically checked on the

running components to distinguish a replaced component from the original one. The idea presented here is based on the timing signature analysis for omega-regular languages [5], in the context of real-time communication scheduling in SCADA systems.

Contributions of this work: In this paper, we address the problem of detecting component replacement attacks from the schedulability perspective. Given a set of control components, a control objective to be satisfied by the control ensemble, the question of schedulability and synthesis of a scheduler that can ensure the desired control performance has been recently studied in literature [5],[6], [7]. In this paper, we extend the same philosophy to build an automata theoretic framework for assessment of replacement attacks on schedulability. The foundation of our attack analysis framework is based on the notion of infinite automata-based reasoning of control performance and schedulability analysis, as illustrated in the following section. Automata theoretic modeling frameworks have been quite popular in literature for a wide variety of applications. Additionally, the power of finite automata over infinite words (Büchi automatons in particular) have been exploited in recent literature in control performance and stability analysis. Our work is another step in the same direction for assessing the effect of replacement attacks in cyber-physical control.

Organization: This paper is organized as follows. Section 2 presents related work. Section 3 describes the problem definition for this work along with a motivating example. Section 4 presents the solution architecture. Section 5 presents an overview of the tool we have developed. Section 6 concludes this discussion.

2 Related Work

Replacement attacks [8] carried out by internal employees, phishing attacks, code injection attacks have been known to be a vector for cyber-attacks for a while. Stuxnet analysis showed that such attacks were part of the repertoire in that case. The idea of automata based control scheduling has been discussed in [9]. They express the communication interface among the control components using a formal language. This work illustrates how the interfaces of discrete-time switched linear systems can be expressed using a Büchi automaton. Authors in [6] describe CPU scheduling in terms of finite machines over infinite words. The infinite words depict the particular time slot for which a particular resource can be allotted to a specific control component. The work in [5] applies automata based scheduling on networks, to formalize the effect of bus scheduling and network stability. Further, [7] describes how the interfacing among the control actions can be modeled by Büchi automata to guarantee stable and reliable scheduling. This work presents a methodology for construction of a scheduler automata where each state represents one particular control schedule while generating the permissible schedule in terms of a ω-regular language. Our work is an application framework based on the contributions above, with specific focus to component

replacement attacks and schedulability analysis. While one of the major control objectives discussed in the above contributions has been exponential stability, which we assume to be satisfied for individual control components, we propose to use a fairness schedule in our work, that requires a different modeling strategy, as explained in the following. More importantly, our work here can be considered as an end-to-end framework on automata-based control performance modeling and assessment.

3 Problem Model

To explain our problem, we consider a control application system consisting of an ensemble of concurrently active control applications / loops. Each control application is partitioned into a set of tasks. The control tasks communicate via a shared bus. The messages on the shared bus are also scheduled using a given arbitration policy. When a control loop executes, it carries out the set of tasks as specified by different states in its control state machine. Among all these states, some states require bus access, for which the control loop presents a request to the bus arbiter. These requests are considered by the bus arbiter / scheduler which decides on a bus scheduling strategy to grant bus access to the different control loops over time at different time slots.

As discussed in recent literature, we consider each participating control application is modeled as a Büchi automaton [10]. Büchi automatons have been a popular mechanism of choice for modeling applications that require finite automatons over infinite strings or words. The basic structure of such automatons is similar to their finite counterparts, with the exception of the acceptance conditions, as described below.

Definition 1. *A Büchi automaton is described as a five tuple, $A = (Q, I, \delta, q_0, F)$, where Q is a finite set of states, I is the input alphabet, $\delta : Q \times I \rightarrow Q$ is the transition function, q_0 is an initial state ($q_0 \in Q$) and F is a set of final states that model an infinitary acceptance condition.* □

Definition 2. *An infinite word $\lambda \in I^{\omega}$ over an input alphabet I takes a Büchi automaton $A = (Q, I, \delta, q_0, F)$ through an infinite sequence of states $q_0, q_1, q_2, ...$, which describes a run of the automaton such that $q_{k+1} \in \delta(q_k, \delta_k)$ where $k \in \mathbb{N}$ and δ_k refers to the k^{th} symbol on the input word δ. An infinite word is accepted by A if some accepting state $q_f \in F$ appears in the run $q_0, q_1, q_2, ...$ infinitely often.* □

Each control loop is designed with a control objective in view, and the overall functioning of the system is dependent on the individual control loops meeting control objectives, along with a strategy for co-operative control of the shared message bus through which the control loops communicate with each other. We consider one such control system functioning correctly if both the above

conditions are met. Meeting control objectives has been addressed in several recent articles, and is therefore, not discussed here. In contrast, we focus more on the schedulability aspect, as described below.

3.1 Schedulability violation attacks:

We now characterize the schedulability requirement below.

Definition 3. *A set of concurrent control loops, each expressed as a Büchi automaton, is schedulable if there exists a strategy to schedule the individual control loops on the shared bus* infinitely often*. The scheduling objective is defined in the lines of infinitary acceptance, as is the case with Büchi objectives.* □

Intuitively, a scheduler should be able to allocate bus access to each individual control loop infinitely often. More specifically, given any infinite run of the composed system, at least one constituent bus accessing / accepting state) from each control loop should repeat infinitely often. We consider a system *safe* if this holds, *unsafe* otherwise. We conclude that a *replacement attack* has been carried out if a system, initially safe and schedulable, turns out to be non-schedulable. We explain our problem statement with a small example below. Fig1 presents two control loops P and Q representing two different control applications. Each has one initial state and an acceptance state. The acceptance states are the bus accessing states of the automaton. Fig 7 shows one possible strategy for scheduling bus access of the two control applications such that their requirements are satisfied. In the diagram, time is plotted along the horizontal axis and the applications are plotted along the vertical axis. The timing diagram depicts how the control applications are scheduled with the bus scheduler. The scheduler gives access to the slot to P control application at every $5th$ occurrence. Other slots are accessed by Q control application.

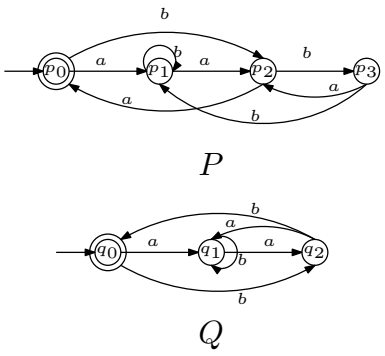

Fig. 1. *Initial Control loops*

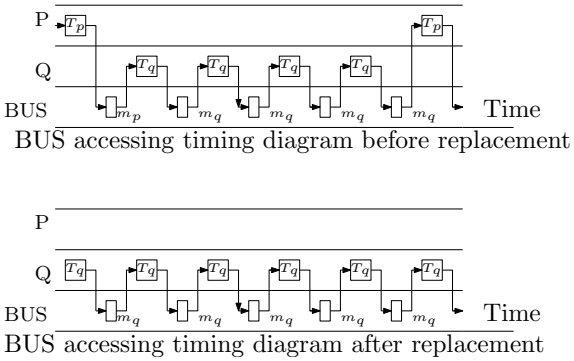

Fig. 2. *Timing diagram of bus scheduler, before and after replacement*

4 Solution Architecture

In this section, we explain our solution architecture for schedulability assessment. As described in the previous section, we are given a set of n control loops, with their bus access states marked as the accepting ones. Recall that our scheduling objective is to be able to grant bus access infinitely often to each control loop. To check if the system is safe and schedulable, we carry out the following steps:

- We compute the product of the individual control loops using the construction explained below.
- Once the product is computed, we look for the presence of a cycle that contains at least one state from each control loop. In other words, on any infinite run of the product automaton, each control loop repeats infinitely often and is therefore, granted bus access infinitely often as well.

4.1 Intersection Automaton construction

Given a collection of control loops, each expressed as Büchi automatons, we describe below the methodology for computing their product [11], which is an important step in our work. For the sake of simplicity and ease of illustration, we explain the product construction in terms of two automatons. Let us consider the two Büchi automatons P and Q, as shown in Fig 3 and defined by the conventional five tuple:

$P = \{(p_0, p_1), (a, b), p_0, Q \times \sum \rightarrow Q, p_1\}$
$Q = \{(q_0, q_1), (a, b), q_0, Q \times \sum \rightarrow Q, q_1\}$

We go through the following steps for the intersection. The classical product on the given input automatons is first constructed. Consider the resulting product automata, starting from the start state. Only the reachable states are considered, as shown in Fig4.

Intersection of Büchi automatons require a special handling to model the infinitary acceptance condition on top of the classical product construction of

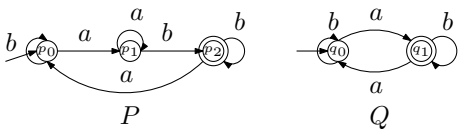

Fig. 3. *Individual automatons*

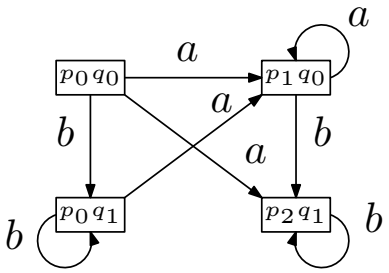

Fig. 4. *Intersection of the input automata*

finite automatons. We can see in Fig 4 that the states do not contain the tuple $< p_2, q_0 >$, the set of final states from both the automata. In the following steps, we modify the product construction to include accepting states from both the automata. The resulting intersection automata ensures that accepting states of both the automatons are visited infinitely often. From the set of states of each individual automaton, where each state is reachable from the start state, we make three copies of the product automata. In general, for n such automatons, n+1 copies are made. Each copy has a flag with the states, as shown in Fig5. The flags associated with the states are waiting to see the acceptance of the automaton. In Fig 5, flag 1 is waiting to see the acceptance state of P, flag 2 is waiting to see the acceptance state of Q, and states with flag 3 signify that the track has traversed all the accepting states of both the automatons. Starting from the first copy, the transitions are made according to the product automaton. A transition goes to the next flag if it gets the final state of one automaton. In general, for an automaton state associated with this flag in the product, if the acceptance condition of an automaton is obtained, the transition goes to the next copy, otherwise it remains within the transitions on its own states, as shown in Fig6. When it reaches the last copy, it has traversed the accepting state of each individual automaton. The red cycle in Fig 6 shows one such cycle. After reaching the last state, it goes to the first state. In the resulting automaton, a cycle is accepted, when it passes through the last copy of the product automaton, that signifies the run of the cycle traverses the accepting states of each automaton infinitely often. The remaining cycles, which do not pass through the last copy, are not accepted. We thus augment the classical product construction for finite automatons to model our infinitary acceptance requirement.

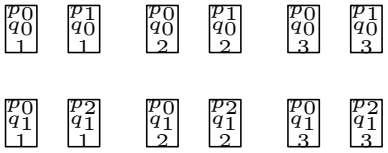

Fig. 5. *Product automata construction with flags*

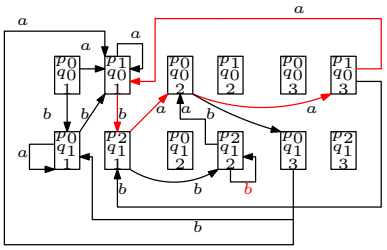

Fig. 6. *Intersection of the input automata*

4.2 Verifying the scheduling objective

Once the intersection automaton is constructed, the task of looking for cycles starting from the initial states is carried out. Cycle detection algorithms are standard in literature [12] and we implement the same for this work as well. Once a cycle is found, we traverse the cycle (every cycle is bound to contain a finite number of states) and check if each control loop is represented inside it. If not, we proceed to the next cycle and carry out the same check. If all cycles are exhausted and we do not encounter any one which meets our scheduling objective, we conclude that the schedulability requirement is not met. This step is thus quite straightforward. Consider the example given in Figure 6. It can be seen that there are multiple cycles in the intersection automata, some of which do not contain one representative accepting state from each control loop (e.g. $< p0, q0, 1 > \rightarrow < p1, q0, 1 > \rightarrow < p0, q1, 1 > \ldots$). Thus it is necessary to examine each cycle to check for existence of at least one accepting state from each control loop, which is the main idea behind the notion of schedulability we adopt here.

4.3 Replacement attack detection

As the system is monitored over time, newer structures of the control loops emerge and we perform the same computation steps outlined above on the modified structures to check if it remains schedulable even in the presence of the

modifications. Our mechanism can thus be used as a continuous monitor to safe-guard against intermittent control attacks.

At this onset, we would like to admit that our framework can point out replacement attacks only if schedulability is lost. Once the intersection is computed, we may have 3 different possibilities if schedulability is lost. Assuming that the system was schedulable earlier, we may conclude the system has been attacked if any of the following possibilities are detected. Firstly, we may have an empty intersection automaton now. As a second possibility, the product may still be non-empty but a cycle may not be reachable. Finally, a cycle may be found but it may not contain representative tasks from each control loop. All these are indicators of schedulability loss according to our analysis method and we conclude that a replacement attack is carried out. However if the system still remains schedulable according to our scheduling objective, our framework is unable to flag the attack, even if a replacement has been carried out.

5 Experimental Results

We have built an end-to-end tool in Python for code replacement attack detection. The tool takes in a set of control loop descriptions, computes their intersection and implements the cycle detection step. The tool has been applied to a number of small examples of synthetic control applications and their variants. The time taken for the tool to run on a QuadCore Intel machine is in the order of milliseconds, and the peak memory consumption is of the order of Kilobytes. Due to the lack of standard open source benchmarks in this domain, we have not been yet able to check the performance of our tool on more non-trivial benchmarks.

We explain our problem statement with a small example. Fig1 presents three automatons P, Q and R representing three different control applications. Each has one initial state and an acceptance state. The acceptance states are the bus accessing states of the automaton. To check for schedulability, we first construct the concurrent composition (the intersection automaton) of the individual automatons, as shown in Fig 7. The product construction is a little different from the classical product of finite automatons, and will be explained in the following section.

We now consider a code replacement attack take places on this system. The resulting automaton is depicted in Fig 8. As a result of this attack, one of the control loops is changed, as a result of which, there is a change in the structure of the intersection automaton. Our schedulability assessment on the new product automaton fails. As we can see in Fig 9, there are no cycles which satisfy our infinitary scheduling condition, thus schedulability is no longer guaranteed and we conclude that a code replacement attack has taken place.

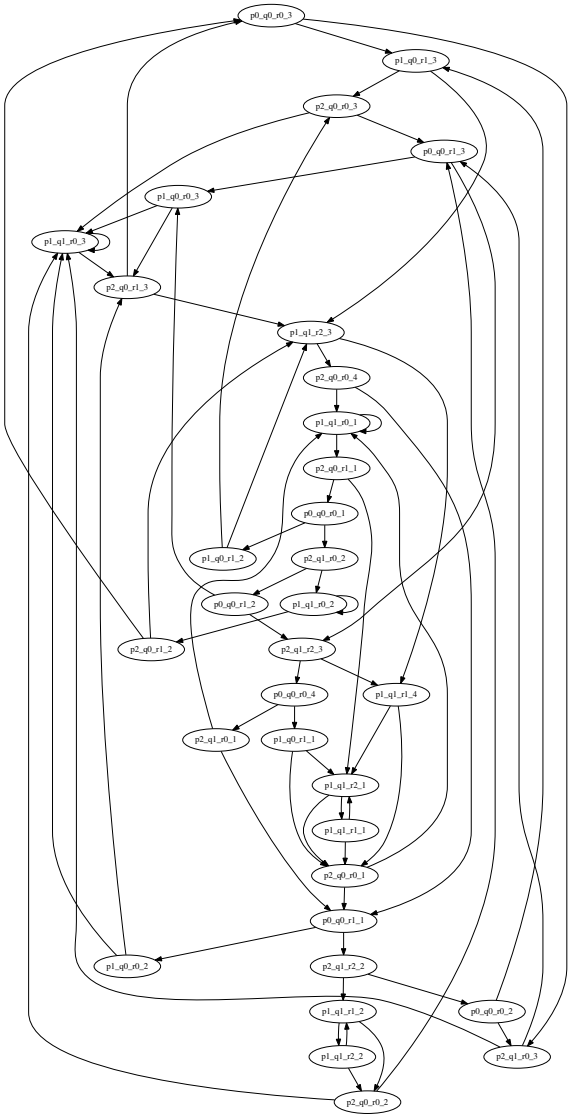

Fig. 7. *Initial Intersection automaton*

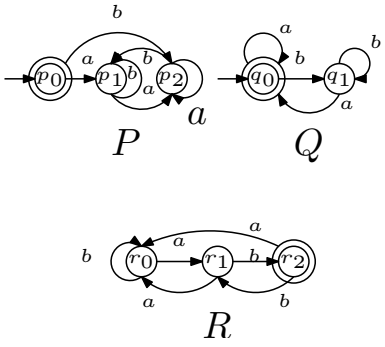

Fig. 8. *Modified control loops after a replacement attack*

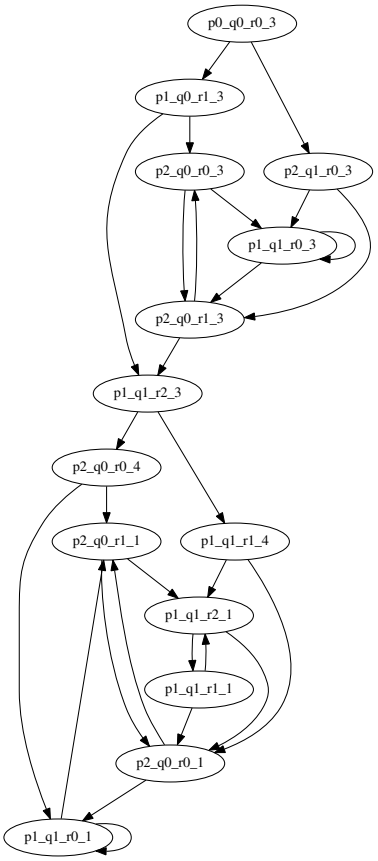

Fig. 9. *Intersection automaton after a code replacement attack*

Table 1. Results

Case	No. of of Statemachines	No. of states in the intersection automata	No. of statesin the cycle	Time taken for product construction	Time taken for cycle construction	Total time taken for sceduling analysis
1	2	8	6	0.00041	0.00082	0.00192
1	2	33	12	0.00029	0.00042	0.00106
3	3	74	15	0.00390	0.00185	0.008127
4	3	149	22	0.00276	0.00390	0.01253
5	5	354	33	0.09007	0.18986	0.30901
6	5	1285	128	0.70243	1.02935	1.83395
7	8	1876	302	69.50649	128.44557	201.55554

6 Conclusion and Future Work

In this work, we present an early foundation of a framework for code replacement attack detection for cyber-physical systems. We adopt standard techniques to build an automata-based control analysis foundation. We have built an end-to-end tool that implements the proposed ideas. We are currently examining the applications of other infinitary acceptance conditions (e.g. Rabin, Muller) in this domain, with respect to their ability to detect different attack scenarios. Additionally, we are also exploring new algorithms for intersection automaton construction that might prove helpful as the number of control loops grow. We believe that our initial results will open up more interesting avenues in this research.

References

1. "Report on the grid disturbances on 30th july and 31st july 2012," National Institute of Standards and Technology, Tech. Rep., August 2012.
2. "National institute of standards and technology." [Online]. Available: http://www.nist.gov
3. "Framework for improving critical infrastructure cybersecurity," National Institute of Standards and Technology, Tech. Rep., February 2014.
4. "The stuxnet worm." [Online]. Available: spectrum.ieee.org/telecom/security/the-real-story-of-stuxnet
5. G. Weiss, S. Fischmeister, M. Anand, and R. Alur, "Specification and analysis of network resource requirements of control systems," in *Proc. of HSSC*, 2009, pp. 381–395.
6. R. Alur and G. Weiss, "Regular specifications of resource requirements for embedded control software," in *Proc. of RTAS*, 2008, pp. 159–168.
7. S. K. Ghosh, A. Mondal, S. Dutta, A. Hazra, and S. Dey, "Synthesis of scheduler automata guaranteeing stability and reliability of embedded control systems," in *Proc. of VDAT*, 2016, pp. 127–132.
8. S. Ghosh, J. Hiser, and J. W. Davidson, "Replacement attacks against vm-protected applications," in *Proc. of VEE*, 2012, pp. 203–214.

9. G. Weiss and R. Alur, "Automata based interfaces for control and scheduling," in *Proc. of HSCC*, 2007, pp. 601–613.
10. W. Thomas, "Automata on infinite objects," in *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, 1990, pp. 133–192.
11. F. Chevalier, D. D'Souza, R. M. Matteplackel, and P. Prabhakar, "Automata and logics over signals," in *Modern Applications of Automata Theory*, 2012, pp. 555–584. [Online]. Available: https://doi.org/10.1142/9789814271059_0018
12. E. M. Clarke, Jr., O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, MA, USA: MIT Press, 1999.
13. M. S. Prabhu, A. Hazra, and P. Dasgupta, "Reliability guarantees in automata-based scheduling for embedded control software," *Embedded Systems Letters*, vol. 5, no. 2, pp. 17–20, 2013.
14. M. S. Prabhu, A. Hazra, P. Dasgupta, and P. P. Chakrabarti, "Handling fault detection latencies in automata-based scheduling for embedded control software," in *Proc. of CACSD*, 2013, pp. 1–6.
15. M. Zamani, S. Dey, S. Mohamed, P. Dasgupta, and M. M. Jr., "Scheduling of controllers' update-rates for residual bandwidth utilization," in *Proc. of FORMATS*, 2016, pp. 85–101.
16. "Sony pictures hack." [Online]. Available: https://en.wikipedia.org/wiki/Sony_Pictures_hack