

An automata theoretic framework for analyzing effects of code replacement attacks on scheduling

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 etc. Usually in ICS, multiple control loops work concurrently and share various resources including communication bus through which they interact with sensors and actuators. The real-time scheduling of concurrent control applications while competing for shared resources, is done with delicate balance between performance and real-time constraints. A possible insider attack could be the replacement of a previously vetted control application or other software components in the system, during a system update. In this paper, we propose an automated framework that addresses the effect of such code 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 the literature. In this paper, we extend this idea further to build an automata theoretic framework for assessment of code replacement attacks on schedulability. We have built an end-to-end framework that takes in a set of control components, their variants (after code replacement), a control objective to be guaranteed, and performs an automated schedulability assessment. We report some preliminary experiments of our framework on simple benchmarks.

Keywords—CyberPhysical System, Cooperative Control, Code Replacement Attack, Insider threat, ω -regular language, Büchi Automaton, Schedulability

I. 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 seem to be the origin areas in the infrastructure.

Research investment in cyber security of CPS is extremely crucial for developing novel technology for cyber attacks 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 popular and widely acknowledged means of cyber-attack, *phishing attacks*, originate when someone whose desktop is connected to the control network opens an email containing a payload, which can then take over the control of the business network, and in turn the control network. However, since much of the cyber threat models also assume insider knowledge or sabotage, even an isolated control network is not devoid of cyber-attack possibilities. A popular example is the notorious Sony Pictures hack [4] which leaked a release of confidential data from the film studio Sony Pictures. 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 some vulnerabilities in their design and attack the cyber physical system. In fact, in case of the Stuxnet worm [5], 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 code replacement attacks, wherein some or parts of a control component are compromised by tampering their operational modality by modifying the software running inside. Code replacement [6] by internal employees, phishing attacks or other means of code injection 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. In an industrial control environment with real time control components, code replacement 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 component. The timing signature analysis of omega-regular languages was originally proposed by researchers in [7], in the context of real-time communication scheduling in SCADA systems.

Contributions of this work: In this paper, we address the problem of detecting code replacement attacks. 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 [7], [8], [9]. In this paper, we extend the same philosophy to build an automata theoretic framework for assessment of code 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 over infinite words (Büchi automata 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 code replacement attacks in cyber-physical control.

The idea of automata based control scheduling has been discussed in [10]. 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. [8] 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 [7] applies automata based scheduling on networks, to formalize the effect of bus scheduling and network stability. Further, [9] describes how the interfacing among the control actions can be modeled by Büchi automaton 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 code replacement attack 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.

Organization: This paper is organized as follows. Section II describes the problem definition for this work along with a motivating example. Section III presents the solution architecture. Section IV presents an overview of the tool and some illustrative examples. Section V concludes this discussion.

II. PROBLEM MODEL AND MOTIVATING EXAMPLE

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 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.

As discussed in recent literature, we consider each participating control application is modeled as a Büchi automaton. Büchi automata [11] have been a popular mechanism of choice for modeling applications that require finite automata over infinite strings or words. The basic structure of such automata is similar to their finite counterparts, with the exception of the acceptance conditions, as described below. 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 help us model the infinitary acceptance condition. An infinite

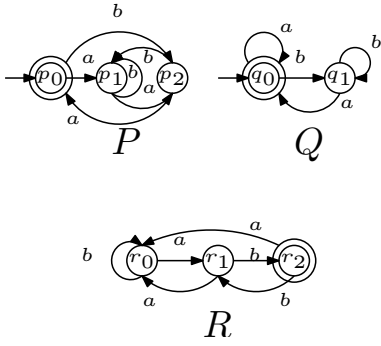


Fig. 1. Initial Control loops

word $\delta \in I^\omega$ over the input alphabet takes the automaton through an infinite sequence of states q_0, q_1, q_2, \dots , 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 if some accepting state $q_f \in F$ appears in the run q_0, q_1, q_2, \dots infinitely often. If $|\delta(q, i)| = 1$ where $q \in Q$ and $i \in I$, then it is a deterministic Büchi automaton, otherwise it is non-deterministic.

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.

A. Schedulability and code replacement attack:

We now characterize the schedulability requirement below.

Definition 2.1: A set of concurrent control loops, each expressed as a Büchi automaton, is considered *schedulable* if there exists a strategy to schedule the individual control loops on the shared bus *infinitely often*. \square

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 task (i.e. bus accessing / accepting state) from each control loop should repeat infinitely often. We consider the system *safe* if this holds, *attacked* otherwise. We conclude that a *code replacement attack* has been carried out if a system, initially safe and schedulable, turns out to be non-schedulable.

B. Motivating Example:

We explain our problem statement with a small example. Fig1 presents three automata P, Q and R representing three different control applications. Each has one initial state and an acceptance state. The acceptance states are the bus

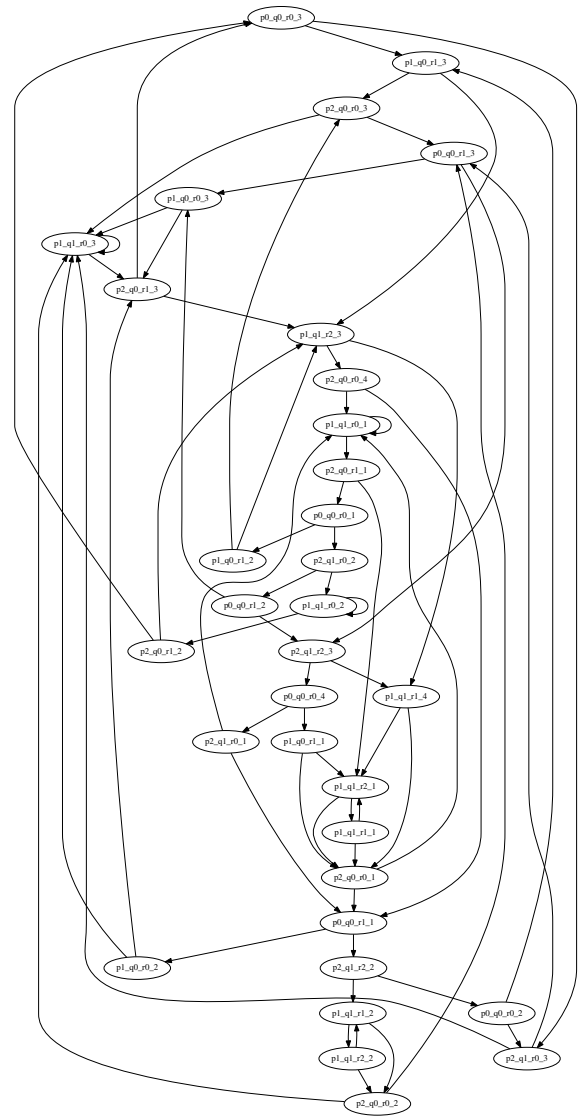


Fig. 2. Initial Intersection automaton

accessing states of the automaton. To check for schedulability, we first construct the concurrent composition (the intersection automaton) of the individual automata, as shown in Fig 2. The product construction is a little different from the classical product of finite automata, 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 3. 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 4, 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.

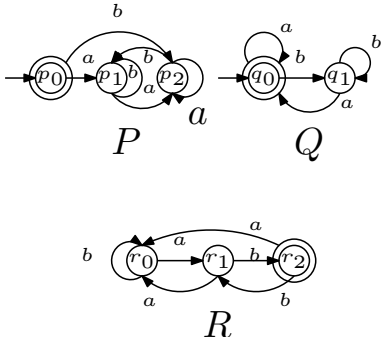


Fig. 3. Modified control loops after a replacement attack

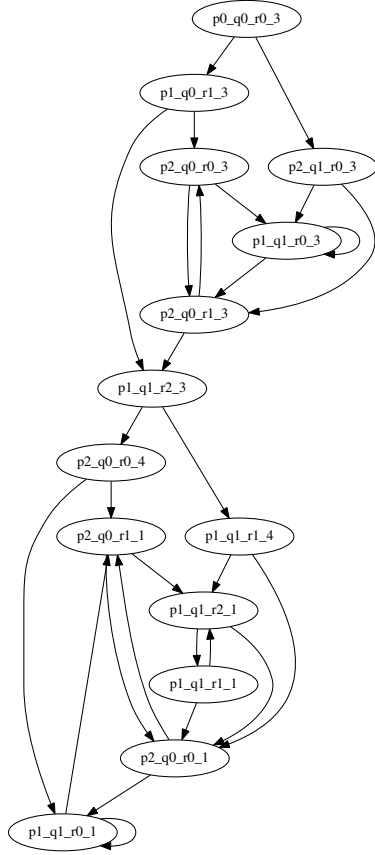


Fig. 4. Intersection automaton after a code replacement attack

III. 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

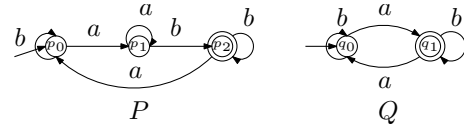


Fig. 5. Individual automata

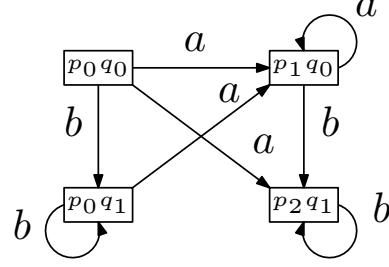


Fig. 6. Cross product of the input automata

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.

A. Intersection Automaton construction

Given a collection of control loops, each expressed as Büchi automata, we describe below the methodology for computing their product [12], 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 automata. Let us consider the two Büchi automata P and Q, as shown in Fig 5 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 cross product on the given input automata is first constructed. Consider the resulting product automata, starting from the start state. Only the reachable states are considered, as shown in Fig6.

Intersection of Büchi automata require a special handling to model the infinitary acceptance condition on top of the classical cross product construction of finite automata. We can see here that the states do not contain the tuple $\langle p_2, q_0 \rangle$, the set of final states from both the automata. In the following steps, we modify the product construction to get a track containing accepting states from both the automata. The resulting intersection automata ensures that accepting states of both the automata 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 automata, $n+1$ copies are made. Each copy has a flag with the states, as shown in Fig7. The flags associated with the states are waiting to see the acceptance of the automaton. In Fig 7, flag 1 is waiting to see the acceptance state of P, flag 2 is waiting to see the acceptance state of Q,

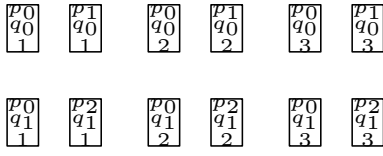


Fig. 7. Product automata construction with flags

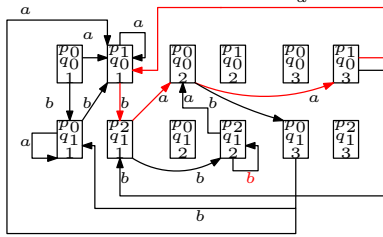


Fig. 8. cross Product of the input automata

and states with flag 3 signify that the track has traversed all the accepting states of both the automata. 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 Fig8. When it reaches the last copy, it has traversed the accepting state of each individual automaton. 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 automata to model our infinitary acceptance requirement.

B. 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 [13] 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.

C. Code Replacement attack detection

As the system is monitored over time, newer structures of the control loops emerge and we perform the same computation

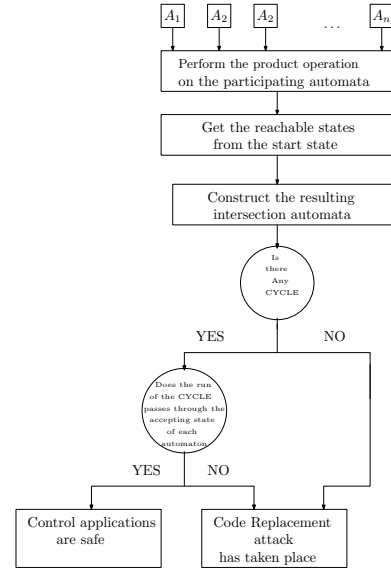


Fig. 9. Sample automata

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 only point out code replacement attacks only if schedulability is lost, not otherwise. Moreover, it may so be the case that a code replacement attack is carried out, however the system still remains schedulable according to our scheduling objective. In such cases as well, our framework is unable to flag the attack.

IV. TOOLFLOW AND CASE STUDY

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 describe below one case study that was analyzed successfully by our tool. An overview of our toolflow is shown in Fig9.

In this section, we have described some examples which depicts the difference of intersection automaton before and after the code replacement. Let us consider three control automata as shown in Fig.10

The intersection automaton of the given automata will look like Fig.11

Now we will see what happens after each of the automaton get replaced.

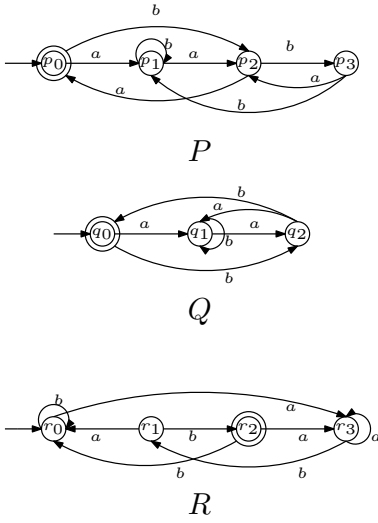


Fig. 10. Sample automata

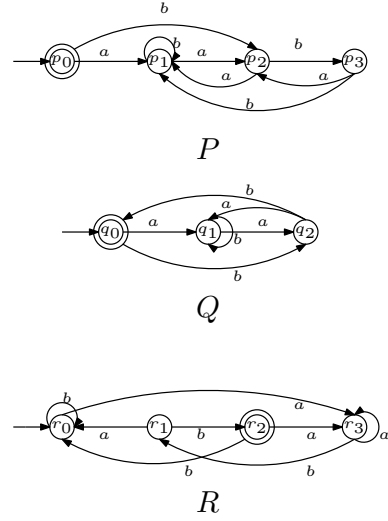


Fig. 12. sample automata: P automaton replaced

Fig. 11. Intersection automata of P, Q and R before replacement

After P automaton replacement the automaton representing the control loops will look like Fig. 12

The intersection automaton will look after P automaton replacement Fig.13

After Q automaton replacement the control loops will look like Fig. 14

The intersection automaton will look after Q automaton replacement Fig.15

After R automaton replacement the control loops will look like Fig. 16

The intersection automaton will look after R automaton replacement Fig.17

V. 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

Fig. 13. Intersection automata of P, Q and R after P got replaced

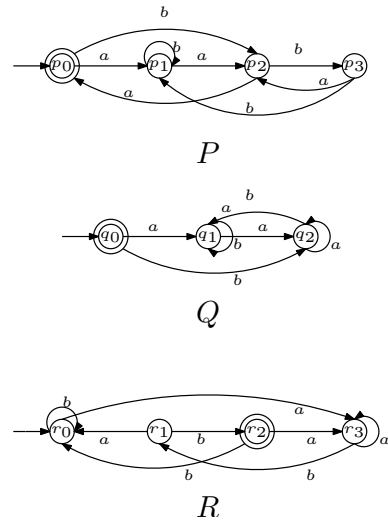


Fig. 14. sample automata: Q automaton replaced

Fig. 15. Intersection automata of P, Q and R after Q got replaced

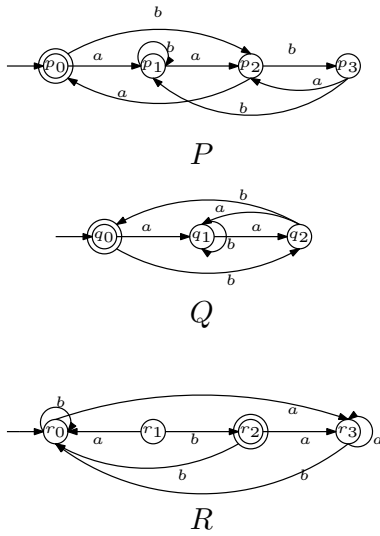


Fig. 16. sample automata: R automaton replaced

Fig. 17. Intersection automata of P, Q and R after R got replaced

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] "Sony pictures hack." [Online]. Available: https://en.wikipedia.org/wiki/Sony_Pictures_hack
- [5] "The stuxnet worm." [Online]. Available: spectrum.ieee.org/telecom/security/the-real-story-of-stuxnet
- [6] S. Ghosh, J. Hiser, and J. W. Davidson, "Replacement attacks against vm-protected applications," in *Proc. of VEE*, 2012, pp. 203–214.
- [7] 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.
- [8] R. Alur and G. Weiss, "Regular specifications of resource requirements for embedded control software," in *Proc. of RTAS*, 2008, pp. 159–168.
- [9] 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.
- [10] G. Weiss and R. Alur, "Automata based interfaces for control and scheduling," in *Proc. of HSCC*, 2007, pp. 601–613.
- [11] W. Thomas, "Automata on infinite objects," in *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B)*, 1990, pp. 133–192.
- [12] 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
- [13] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, MA, USA: MIT Press, 1999.
- [14] 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.
- [15] 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.

- [16] 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.