On Security Analysis of Periodic Systems: Expressiveness and Complexity

Musab A. Alturki[1,2], Tajana Ban Kirigin[3], Max Kanovich[4,5], Vivek Nigam[6], Andre Scedrov[7] and Carolyn Talcott[8]

[1]*KFUPM, Dhahran, Saudi Arabia*

[2]*Runtime Verification Inc., U.S.A.*

[3]*Department of Mathematics University of Rijeka, Rijeka, Croatia*

[4]*University College London, London, U.K.*

[5]*National Research University Higher School of Economics, Moscow, Russian Federation*

[6]*fortiss, Munich, Germany*

[7]*University of Pennsylvania, Philadelphia, PA, U.S.A.*

[8]*SRI International, Menlo Park, CA, U.S.A.*

Keywords: Formal Methods, Verification, Security, Multiset Rewriting, Industry 4.0, Complexity.

Abstract: Development of automated technological systems has seen the increase in interconnectivity among its components. This includes Internet of Things (IoT) and Industry 4.0 (I4.0) and the underlying communication between sensors and controllers. This paper is a step toward a formal framework for specifying such systems and analyzing underlying properties including safety and security. We introduce automata systems (AS) motivated by I4.0 applications. We identify various subclasses of AS that reflect different types of requirements on I4.0. We investigate the complexity of the problem of functional correctness of these systems as well as their vulnerability to attacks. We model the presence of various levels of threats to the system by proposing a range of intruder models, based on the number of actions intruders can use.

1 INTRODUCTION

Technologies such as networked devices including simple sensors and controllers as well as cyber-physical systems and Internet of Things (IoT) are being increasingly adopted in industry to improve production efficiency and to enable process agility and product personalization. This trend is referred to as Industry 4.0 (I4.0). The combination of flexible interconnectivity and insecure devices also presents opportunities for cyber-attacks. In an industrial setting such attacks lead to serious material or human damage. One example is the attack on a steel mill, requiring the factory to stop its production resulting in great financial loss (Cyberattack, 2015).

The IEC 61499 international standard for distributed industrial control systems (Zoitl and Lewis, 2014; Yoong et al., 2015) proposes defining the functionality of the whole system using a platform-independent Application model. The model is composed of elements called *function blocks* (FBs) that interact via data and event interfaces (Zoitl and Lewis, 2014; Yoong et al., 2015).

There is a number of works carrying out systematic, but informal security analysis for I4.0 systems including a recent BSI report on the security of OPC-UA (machine to machine communication protocol for industrial automation) (Fiat et.al., 2017) and the ENISA study on good practices for IoT security (ENISA, 2018).

This paper is a step toward a formal analysis of I4.0 applications. Our formal framework is based on Multiset Rewriting (Durgin et al., 2004) (MSR). Motivated by the requirements of I4.0 applications, we propose different MSR models:

Automata Systems (AS) are systems similar to those specified by the IEC 61499 standard. In particular, FBs are specified as possibly non-deterministic Mealy machines (Savage, 1998) that interact by carrying out local computations and exchanging events.

Alturki, M., Kirigin, T., Kanovich, M., Nigam, V., Scedrov, A. and Talcott, C.
On Security Analysis of Periodic Systems: Expressiveness and Complexity.
DOI: 10.5220/0010195100430054
In *Proceedings of the 7th International Conference on Information Systems Security and Privacy (ICISSP 2021)*, pages 43-54
ISBN: 978-989-758-491-6

Table 1: Summary of complexity results for Functional Correctness Problem (FCP) and Security Problem for Functionally Correct Systems against Intruders (SP-FCS).

System Model	FCP	SP-FCS	SP-FCS with an Intruder using only one action
AS	PSPACE-Complete [Th.4.11, Th.4.12]	In PSPACE [Th.5.5]	PSPACE-Hard [Th.5.5]
PAS	PSPACE-Complete [Th.4.11, Th.4.12]	In PSPACE [Th.5.5]	PSPACE-Hard [Th.5.5]
LAS	coNP-Complete [Th.4.13, Th.4.14]	In PSPACE [Th.5.5]	PSPACE-Hard [Th.5.5]

Periodic Automata Systems (PAS) refine AS by incorporating the assumption that I4.0 applications are periodic. That is, an application carries out a collection of tasks by execution of its FBs periodically.

Locally Bounded Periodic Automata Systems (LAS) refine PAS by bounding the number of executions of each FB withing one system cycle.

We first investigate the complexity of the *Functional Correctness Problem* (FCP), that is, deciding whether a system does not lead to a critical configuration that may lead to human or financial losses. FCP can be seen as checking whether the system behaves correctly without the presence of an intruder.

When considering an intruder, we follow the findings of the BSI report on OPC-UA security (Fiat et.al., 2017). The report concludes that message injection and tampering attacks pose the most serious threats to I4.0 applications. Following this assessment, we propose intruder models, inspired by the Dolev-Yao intruder model (DY) (Dolev and Yao, 1983), where the intruder controls the network. Our intruders can inject, tamper and block messages. We also consider a bounded version of intruders that can only interfere with the system a bounded number of times.

We then investigate the Security Problem for Functionally Correct Systems (SP-FCS), that is, determining whether a functionally correct system can reach a critical configuration in the presence of an intruder. Obtained results are summarized in Table 1. Our computational complexity results refer to standard complexity classes NP (non-deterministic polynomial time) and PSPACE (polynomial space) (Savage, 1998).

Even with the relatively simple AS model, the complexity of both problems is PSPACE-complete. A class of AS, LAS, for which the complexity of FCP is co-NP-complete is identified. However, the complexity of SP-FCS does not improve, even in the case when the intruder is allowed to use only one action.

Sections 2 and 3 motivate this work with related work and an example taken from an I4.0 application. In Section 4 we introduce AS as a MSR model and specify various classes of AS. We define FCP proving complexity results for various AS classes. In Section 5, we introduce MSR intruder models and present complexity results for SP-FCS for different assumptions on intruders and types of systems. In Section 6 we present results of SP-FCS experiments obtained using Maude. We conclude in Section 7 by pointing to future work. Appendix contains the proofs of the complexity results.

2 RELATED WORK

Recently, (Lanotte et al., 2020) proposed methods for the verification of cyber-physical systems, taking into account their actual physical behavior. In contrast, our approach does not enter into such details, but rather only considers an abstract level. This greatly affects the type of verification that is done. (Lanotte et al., 2020) uses statistical model checking approaches, while we use a symbolic approach from (Nigam and Talcott, 2019). That approach combines formal executable specification of I4.0 applications with a bounded intruder model by means of rewrite modules in Maude (Clavel et al., 2007). Such bounded intruder is already capable of causing damage by injecting system messages to be received at the wrong time, causing safety invariants to be violated and a bad state to be reached. For an equationally defined bad state, all attack scenarios can be enumerated using Maude's search capability. The symbolic approach may be combined with abstraction techniques such as (Nigam and Talcott, 2020). Such abstraction techniques support the engineering design workflow using theory transformations. In particular, given a deployment map from application components to devices, one can define a theory transformation that models execution of the application on the given set of (networked) devices. Given an enumeration of attacks (message flows) one can further define a theory transformation that provides a security wrapper for each device with policies for signing/signature checking for just those messages needed to prevent the attacks (Nigam and Talcott, 2020). This paper provides a mathematical foundation for the specification framework in (Nigam and Talcott, 2019), which is executable in Maude.

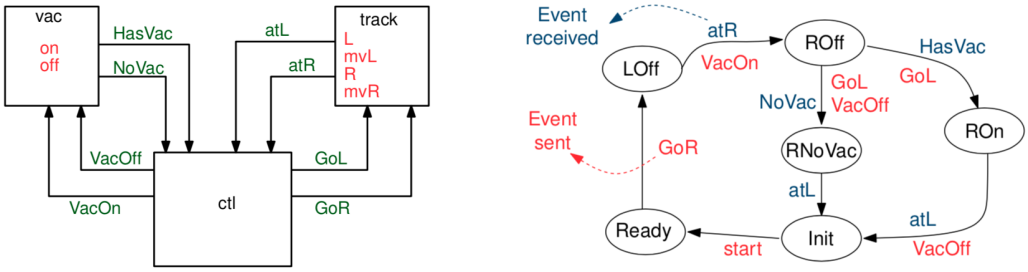


Figure 1: PnP Function Blocks, ctl, vac, and track. The internal states of vac and track are shown in their corresponding boxes and their transitions are elided. The complete specification of ctl is shown in the finite machine to the right.

3 MOTIVATING EXAMPLE

Figure 1 shows the application architecture of a simple I4.0 unit, called Pick and Place (PnP)[1]. This is a common pattern in production lines where an arm *pick*s up something from one location and *place*s it at another. In this example there is a conveyor belt that brings containers to a barrier point, and a source of caps. The arm moves along a track to the right positioning over the caps. It turns on its vacuum mechanism to lift a cap and moves along the track to the left where it is positioned over a container. It turns off the vacuum and is ready for the next cycle.

The PnP application model consists of three function blocks: track (controls the movement of the arm); vac (controls the vacuum on/off state and contains a sensor for detecting whether the piece has been picked up); and ctl (coordinates the track and vac FBs). These FBs communicate according to the links shown in Figure 1 on the left. The behavior of a FB is specified by an interactive automata similar to a Mealy automata. Transitions are guarded by predicates on incoming signals (called events). When a transition fires, outgoing events are generated and transformed into incoming events according to the network links.

The automata for ctl is shown on the right in Figure 1. From its initial state, Init, ctl sends to itself the message start to start a new production cycle migrating to state Ready. It then transitions to LOff state, representing the system with the arm at the left end and the vacuum off. It emits a GoR event that instructs track to move the arm to the right position and confirm that by an atR event. When an event atR arrives, ctl transitions to ROff state, denoting that the arm is at the right and the vacuum is off. It emits VacOn to turn on the vacuum. The controller proceeds in a similar fashion taking the cap from the right side to the left, bringing the cap to the correct position. Then it places the cap over the cylinder by deactivating the vacuum. If vac fails to pick a cap, it sends NoVac event to ctl. In this case, ctl moves to state RNoVac and sends an event to de-activate vac and move to the left side of the PnP. This is a typical manufacturing application where a task is repeated periodically. This is reflected in the fact that all cycles in the specification of FBs, include the initial state.

For ensuring the safety of the system, analysis of the logical behavior of the system is normally carried out using methods such as Systems Theoretic Process Analysis (STPA). The analysis should determine which are the bad system configurations that should be avoided as they pose safety hazards.

For example, a safety hazard for the PnP is the event of a cap falling while being moved. (Imagine instead of putting caps on containers the PnP unit is placing heavy bricks.) Releasing the piece prematurely could injure someone that is passing by or damage the factory itself, *e.g.*, damaging the conveyor belt. This safety hazard is related to the *critical configuration* where track is in state mvL, vac is in off state and ctl in ROn or in Init state. That is, ctl has received the signal that vac has picked the cap, but while moving to the left, the state of vac is off, indicating that the cap has been released. One way a bad configuration could be reached is if ctl sends a VacOff event before the arm is all the way to the left (maybe trying to optimize something). This would constitute an I4.0 application that is not *functionally correct*, as it is possible to reach the critical configuration above.

Additionally, if the application is functionally correct, we ask if an intruder that is capable of injecting an event into any one of the links, at any time, can drive PnP into a critical configuration. The answer is yes. Indeed, as described in (Nigam and Talcott, 2019), there are four ways in which the intruder can do this. For example, while the cap is being moved, the attacker can inject the message VacOff to vac, causing it to release the cap. Alternatively, attacker

[1]See https://www.youtube.com/watch?v=Tkcv-mbhYqk starting at 55 seconds for a very small scale version of the PnP

may send the message atL to ctl, although the cap is still being moved, thus causing the ctl to prematurely deactivate the vac and release the cap.

4 FORMAL MODEL

We briefly review MSR (Kanovich et al., 2014) which is the language we use to specify systems and intruders. Assume a finite first-order typed alphabet, Σ, with variables, constants, function and predicate symbols. Terms and facts are constructed by applying symbols with correct type. For instance, if P is a predicate of type $\tau_1 \times \tau_2 \times \cdots \times \tau_n \to o$, where o is the type for propositions, and $u_1, \ldots, u_n$ are terms of types $\tau_1, \ldots, \tau_n$, respectively, then $P(u_1, \ldots, u_n)$ is a *fact*. A *configuration* is a multiset of ground facts.

Actions are multiset rewrite rules:

$W_1, \ldots, W_k, F_1, \ldots, F_n \longrightarrow W_1, \ldots, W_k, Q_1, \ldots Q_m$

used to change configurations, *i.e.*, model processes. Facts $W_1, \ldots, W_k$ are preserved by the above rule, while facts $F_1, \ldots, F_n$ are replaced by $Q_1, \ldots, Q_m$. All free variables appearing in the post-condition must appear in the pre-condition. A rule of the form $\mathcal{W} \longrightarrow \mathcal{W}'$ can be applied to a configuration $\mathcal{S}$ if there is a subset $\mathcal{S}_0 \subseteq \mathcal{S}$ and a matching substitution θ, such that $\mathcal{S}_0 = \mathcal{W}\theta$. The configuration resulting from the application of this rule to $\mathcal{S}$ is $(\mathcal{S} \setminus \mathcal{S}_0) \cup (\mathcal{W}'\theta)$. (Substitution application $(\mathcal{S}\theta)$ is defined by mapping term variables to terms.)

Definition 4.1 (Trace)**.** *A* trace *of MSR rules $\mathcal{R}$ is a sequence of configurations:*

$\mathcal{S}_0 \longrightarrow_{r_1} \mathcal{S}_1 \longrightarrow_{r_2} \cdots \longrightarrow_{r_n} \mathcal{S}_n \longrightarrow_{r_{n+1}} \ldots$

or its finite prefix, such that for all $0 \leq i$, $\mathcal{S}_{i+1}$ is a configuration obtained by applying $r_{i+1} \in \mathcal{R}$ to $\mathcal{S}_i$.

Reachability problems are reduced to the existence of traces over given rules from some initial to some specified configuration. Since reachability problems are undecidable in general (Kanovich et al., 2011), some restrictions are imposed in order to obtain decidability. In particular, we only use MSR systems with *balanced* rules, *i.e.*, rules for which the number of facts in its pre-condition and in its post-condition is the same. Systems containing only balanced rules represent an important class of *balanced systems*, for which several reachability problems have been shown decidable (Kanovich et al., 2011; Kanovich et al., 2014; Kanovich et al., 2017).

Our MSR systems representing I4.0 application and intruders are balanced, denoting a fixed setting of function blocks communicating using a fixed set of signals through a network of a fixed capacity.

4.1 Industry 4.0 Specifications

We now present how systems described in Section 3, are specified as formal MSR models. We use the signature containing a finite number of constants denoting signals, constants denoting automata states, predicates denoting states of each automaton, and predicates used to denote channels, respectively.

We define *automata systems* representing a network of FBs which are conceived of as the event-driven finite automata A_1, A_2, ..., A_ℓ. Some of the automata, say A and B, can directly interact through a channel on which A has the write access and B has the read/consume access. We denote such a channel with the predicate $R_{A,B}$. Given a channel $R_{A,B}$, the fact $R_{A,B}(\mathtt{m})$ denotes that, via $R_{A,B}$, A provides an event-driven signal m to be consumed in some moment by the intended recipient B, while $R_{A,B}(*)$ denotes that the channel is empty. An interpretation is that $R_{A,B}$ is a one-cell buffer that may contain a "signal".

Definition 4.2 (Automata System)**.** *An* automata system (AS) *is a pair $\mathcal{N} = (\mathcal{A}, \mathcal{R})$, where $\mathcal{A} = \{A_1, \ldots, A_n\}$ is a finite set of automata and $\mathcal{R}$ a finite set of channels $\mathcal{R}_{A_i,A_j}$ from A_i to A_j, $A_i, A_j \in \mathcal{A}$, such that for any pair of channels $R_{A_i,A_j}, R_{A_l,A_k} \in \mathcal{R}$ if $R_{A_i,A_j} = R_{A_l,A_k}$, then $i = l$ and $j = k$.*

An automaton *A of AS $\mathcal{N} = (\mathcal{A}, \mathcal{R})$ is a tuple (S_A, q_0, M_A, X_A), where S_A is a finite set of automaton states with an initial state $q_0 \in S_A$, M_A a finite set of message symbols not containing the $*$ symbol, and X_A a finite set of* instructions *of the form:*

$$Q_A(\mathtt{q}), R_{B_1,A}(\mathtt{m}_1), \ldots, R_{B_k,A}(\mathtt{m}_k), R_{A,C_1}(*), \ldots, R_{A,C_\ell}(*) \longrightarrow Q_A(\mathtt{q}'), R_{B_1,A}(*), \ldots, R_{B_k,A}(*), R_{A,C_1}(\mathtt{m}'_1), \ldots, R_{A,C_\ell}(\mathtt{m}'_\ell) \quad (1)$$

where $\mathtt{m}_1, \ldots, \mathtt{m}_k, \mathtt{m}'_1, \ldots, \mathtt{m}'_\ell \in M_A$, and $\mathtt{q}, \mathtt{q}' \in S_A$.

Automata in an AS communicate by exchanging a fixed set of (atomic) signals through a fixed number of distinct channels. Each of the automaton A is defined by a finite set of balanced rules representing how the received event signals prompt automata to action. We refer to all instructions of automata in $\mathcal{N} = (\mathcal{A}, \mathcal{R})$ as *system rules* $X_\mathcal{N}$, that is $X_\mathcal{N} = \bigcup_{A \in \mathcal{A}} X_A$.

As per rule Eq. (1), when each of the channels R_{A,C_j} is free, automaton A being in the state q, getting the signals $\mathtt{m}_i$ via channels $R_{B_i,A}$, respectively, moves to its state q', provides the signals $\mathtt{m}'_j$ via channels R_{A,C_j}, respectively, and in the process discharges the signals $\mathtt{m}_j$, freeing all channels $R_{B_i,A}$. In the special case when $k = 0$, rule Eq. (1) denotes automaton action prompted internally, only by automaton state.

For an example, consider the PnP system illustrated in Figure 1 and the corresponding rules given in Figure 2. Within this system, the instruction $\mathtt{c}_3$ denotes the following action of ctl automata: Being in

$$
\begin{aligned}
\textsf{ctl}\ \mathtt{c}_1 &: Q_{\textsf{ctl}}(\textsf{Init}), R_{\textsf{ctl,ctl}}(*) \rightarrow Q_{\textsf{ctl}}(\textsf{Ready}), R_{\textsf{ctl,ctl}}(\textsf{start})\\
\mathtt{c}_2 &: Q_{\textsf{ctl}}(\textsf{Ready}), R_{\textsf{ctl,ctl}}(\textsf{start}), R_{\textsf{ctl,track}}(*) \rightarrow\\
&\quad Q_{\textsf{ctl}}(\textsf{LOff}), R_{\textsf{ctl,ctl}}(*), R_{\textsf{ctl,track}}(\textsf{GoR})\\
\mathtt{c}_3 &: Q_{\textsf{ctl}}(\textsf{LOff}), R_{\textsf{track,ctl}}(\textsf{atR}), R_{\textsf{ctl,vac}}(*) \rightarrow\\
&\quad Q_{\textsf{ctl}}(\textsf{ROff}), R_{\textsf{track,ctl}}(*), R_{\textsf{ctl,vac}}(\textsf{VacOn})\\
\mathtt{c}_4 &: Q_{\textsf{ctl}}(\textsf{ROff}), R_{\textsf{vac,ctl}}(\textsf{HasVac}), R_{\textsf{ctl,track}}(*) \rightarrow\\
&\quad Q_{\textsf{ctl}}(\textsf{ROn}), R_{\textsf{vac,ctl}}(*), R_{\textsf{ctl,track}}(\textsf{GoL})\\
\mathtt{c}_5 &: Q_{\textsf{ctl}}(\textsf{ROn}), R_{\textsf{track,ctl}}(\textsf{atL}), R_{\textsf{ctl,vac}}(*) \rightarrow\\
&\quad Q_{\textsf{ctl}}(\textsf{Init}), R_{\textsf{track,ctl}}(*), R_{\textsf{ctl,vac}}(\textsf{VacOff})\\
\mathtt{c}_6 &: Q_{\textsf{ctl}}(\textsf{ROff}), R_{\textsf{vac,ctl}}(\textsf{NoVac}), R_{\textsf{ctl,track}}(*), R_{\textsf{ctl,vac}}(*) \rightarrow\\
&\quad Q_{\textsf{ctl}}(\textsf{RNoVac}), R_{\textsf{vac,ctl}}(*), R_{\textsf{ctl,track}}(\textsf{GoL}), R_{\textsf{ctl,vac}}(\textsf{VacOff})\\
\mathtt{c}_7 &: Q_{\textsf{ctl}}(\textsf{RNoVac}), R_{\textsf{track,ctl}}(\textsf{atL}), R_{\textsf{ctl,ctl}}(*) \rightarrow\\
&\quad Q_{\textsf{ctl}}(\textsf{Init}), R_{\textsf{track,ctl}}(*), R_{\textsf{ctl,ctl}}(\textsf{start})\\
\textsf{track}\ \mathtt{t}_1 &: Q_{\textsf{track}}(\textsf{L}), R_{\textsf{ctl,track}}(\textsf{GoR}) \rightarrow Q_{\textsf{track}}(\textsf{mvR}), R_{\textsf{ctl,track}}(*)\\
\mathtt{t}_2 &: Q_{\textsf{track}}(\textsf{mvR}), R_{\textsf{track,ctl}}(*) \rightarrow Q_{\textsf{track}}(\textsf{R}), R_{\textsf{track,ctl}}(\textsf{atR})\\
\mathtt{t}_3 &: Q_{\textsf{track}}(\textsf{R}), R_{\textsf{ctl,track}}(\textsf{GoL}) \rightarrow Q_{\textsf{track}}(\textsf{mvL}), R_{\textsf{ctl,track}}(*)\\
\mathtt{t}_4 &: Q_{\textsf{track}}(\textsf{mvL}), R_{\textsf{track,ctl}}(*) \rightarrow Q_{\textsf{track}}(\textsf{L}), R_{\textsf{track,ctl}}(\textsf{atL})\\
\textsf{vac}\ \mathtt{v}_1 &: Q_{\textsf{vac}}(\textsf{off}), R_{\textsf{ctl,vac}}(\textsf{VacOn}), R_{\textsf{vac,ctl}}(*) \rightarrow\\
&\quad Q_{\textsf{vac}}(\textsf{on}), R_{\textsf{ctl,vac}}(*), R_{\textsf{vac,ctl}}(\textsf{HasVac})\\
\mathtt{v}_2 &: Q_{\textsf{vac}}(\textsf{off}), R_{\textsf{ctl,vac}}(\textsf{VacOn}), R_{\textsf{vac,ctl}}(*) \rightarrow\\
&\quad Q_{\textsf{vac}}(\textsf{on}), R_{\textsf{ctl,vac}}(*), R_{\textsf{vac,ctl}}(\textsf{NoVac})\\
\mathtt{v}_3 &: Q_{\textsf{vac}}(\textsf{on}), R_{\textsf{ctl,vac}}(\textsf{VacOff}), R_{\textsf{vac,ctl}}(*) \rightarrow\\
&\quad Q_{\textsf{vac}}(\textsf{off}), R_{\textsf{ctl,vac}}(*), R_{\textsf{vac,ctl}}(*)
\end{aligned}
$$

Figure 2: Instructions of PnP AS.

the state LOff and getting the signal atR denoting that the arm is in the right-most position, ctl sends the signal VacOn to engage the vacuuming action with the vacuum device. Rules $\mathtt{c}_1, \mathtt{c}_2, \mathtt{c}_3, \mathtt{c}_4, \mathtt{c}_5$ formalize one cycle of the automaton ctl, while rules $\mathtt{c}_1, \mathtt{c}_2, \mathtt{c}_3, \mathtt{c}_6, \mathtt{c}_7$ formalize the other cycle illustrated in Figure 1.

Definition 4.3 (System Configuration)**.** *Given an AS $\mathcal{N} = (\mathcal{A}, \mathcal{R})$, a* system configuration *of $\mathcal{N}$ is a multiset of facts containing exactly one fact $Q_{A_i}(q)$, for each $A_i \in \mathcal{A}$, where $q \in S_{A_i}$, and exactly one fact $R_{A_i,A_j}(\mathtt{m})$, for each $R_{A_i,A_j} \in \mathcal{R}$, where $\mathtt{m} \in M_{A_i}$.*

A system configuration represents a snapshot of the AS, containing the current states of all automata and the contents of the connecting channels. Notice that, since exactly one of each channel predicates R_{A_i,A_j} is included, we model systems with at most one channel from one automata in the system to another, each channel of a single buffer capacity.

The assumption of single buffer capacity normally appears in many I4.0 applications, in particular, for (parts of) applications that require high performance or are safety critical (Ademaj et al., 2019). This is implemented by using message delivery schedules, such as those in Time Sensitive Networks, so that only a single message is received and processed at a time. It is possible, however, to extend our model so that multiple channels and larger network capacities can be represented, *e.g.*, by using multiple $R_{A,B}$ facts in the configuration or by using special facts denoting network bandwidth. However, the implications of such extensions on the complexity is left for future work.

Definition 4.4 (Initial Configuration)**.** Initial configuration *of an AS $\mathcal{N} = (\mathcal{A}, \mathcal{R})$ is the system configuration of $\mathcal{N}$ with one $Q_{A_i}(q_0^i)$ fact, for each $A_i \in \mathcal{A}$, where q_0^i is the initial state of A_i, and one fact $\mathcal{R}_{A_i,A_j}(*)$ for each channel $\mathcal{R}_{A_i,A_j} \in \mathcal{R}$.*

Definition 4.5 (Critical Configuration)**.** *Given an AS, we assume a set of system configurations called* critical configurations*. We also assume the existence of a polynomial time algorithm C that recognizes which system configuration is critical and which is not.*

Critical configurations denote bad overall configuration of the system. For example, for the system illustrated in Figure 1, it may be critical that the vacuum switches off while the arm is moving left, carrying a cap. Then, any configuration containing either the facts $\{Q_{\textsf{vac}}(\textsf{off}), Q_{\textsf{track}}(\textsf{mvL}), Q_{\textsf{ctl}}(\textsf{ROn})\}$ or $\{Q_{\textsf{vac}}(\textsf{off}), Q_{\textsf{track}}(\textsf{mvL}), Q_{\textsf{ctl}}(\textsf{Init})\}$ would be critical.

We assume that each AS has an associated specification of critical configurations. Such configurations represent situations that are undesired w.r.t. functionality of the I4.0 application being modelled by the AS.

Given that I4.0 applications are written as Mealy machines, one could question the motivation of using MSR models. One reason is that it is straightforward to add intruder models as we describe in Section 5 and define the corresponding verification problems. Mealy machines are not suitable for specifying intruders that can send messages at any time in any one of the channels. Another reason is that MSR rules are more general and can be used to express further features, such as nonces used in protocol security, that are not available in Mealy machines. While nonces and cryptographic protocols, in general, are not used in this paper, our models can easily be extended to formalize *e.g.*, signed messages.

4.1.1 Periodic Automata Systems

We introduce subclasses of AS by incorporating further requirements of I4.0. A typical I4.0 application is periodic, that is, a collection of tasks are repeated over and over again. For example, the PnP described in Section 3, repeats the task of placing a cap over a cylinder. In I4.0 terminology, FBs operate in micro-cycles, where each automaton repeats one of its cycles, while the whole application operates in hyper-cycles, which start and end in a system configuration where all FBs are in their initial states.

Definition 4.6 (Hyper-Cycle)**.** *Let $\mathcal{N} = (\mathcal{A}, \mathcal{R})$ be an AS. A* hyper-cycle *of $\mathcal{N}$ is a trace of sytem rules $X_{\mathcal{N}}$, $\mathcal{S}_0 \longrightarrow_{r_1} \mathcal{S}_1 \longrightarrow_{r_2} \cdots \longrightarrow_{r_n} \mathcal{S}_n$, $n \geq 1$, where $\mathcal{S}_I$ is the initial configuration of $\mathcal{N}$, $\mathcal{S}_0 = \mathcal{S}_n = \mathcal{S}_I$, $\mathcal{S}_i \neq \mathcal{S}_I$, and $\mathcal{S}_i \neq \mathcal{S}_j$, $\forall i, j \in \{1, \ldots n-1\}$.*

To model periodic behavior, we introduce a class of systems called Periodic Automata Systems (PAS) which imposes constraints on the system behaviour.

Definition 4.7 (Periodic Automata System)**.** *An AS $\mathcal{N}$ is* periodic (PAS) *if any finite trace of $\mathcal{N}$ starting from the initial configuration of $\mathcal{N}$ is a prefix of an infinite trace, and any infinite trace of $\mathcal{N}$ starting from the initial configuration of $\mathcal{N}$ is the concatenation of its hyper-cyles.*

For example, the PnP in Figure 1 is a PAS. In particular, ctl may run in two different cycles through its initial state Init, the outer one and the inner one.

Proposition 4.8. *Given a PAS, a system configuration is reachable from an initial configuration if and only if it is reachable within one hyper-cycle.*

In a PAS the number of applications of instructions within any hyper-cycle of any automaton could in principle be exponential. On the other hand, notice that in the PnP example in Figure 1 each of the instructions is applied at most once in a hyper-cycle.

Definition 4.9 (Locally Bounded Periodic Automata System)**.** *A PAS $\mathcal{N} = (\mathcal{A}, \mathcal{R})$, where $\mathcal{A} = \{A_1, \ldots, A_n\}$, is* k-bounded *if the number of applications of instructions of any A_i within a hyper-cycle of $\mathcal{N}$ is at most k. A PAS is* locally bounded (LAS) *if it is k-bounded for some explicitly given k.*

4.2 Functional Correctness

Functional correctness is an unreachability problem with critical configurations specified over states of FBs, denoting bad configurations of the system.

Definition 4.10 (Functional Correctness (FCP))**.** *An automata system $\mathcal{N}$ is said to be* functionally correct *if there is no trace of $\mathcal{N}$ leading from the initial configuration of $\mathcal{N}$ to a critical configuration.*

FCP is a safety property for AS. Functionally correct systems guarantee correct execution of the working process, within the closed system with no outside interference. However, this does not guarantee security, as intruder actions may lead to undesired system configurations.

For AS, in general, the complexity of FCP is high. Traces and even hyper-cycles may be of exponential length. Namely, the number of different system configurations is bounded by $s^n \cdot m^c$, where n is the number of automata in the system, c is the number of channels, s is the bound on the number of states of any automaton, and m is the bound on the number of different messages that can be sent on any channel. This number is exponential in the number of automata and channels in the system.

Theorem 4.11. *FCP for AS is in PSPACE.*

Proof Sketch. We take into account that the number of channels and their capacity are supposed to be fixed in advance. Any intermediate configuration that includes the states of automata and the contents of the interface channels is of polynomial size. Therefore, the existence of an appropriate sequence of actions from the initial configuration can be guessed in PSPACE (Kanovich et al., 2011). Functional correctness can be done in co-PSPACE. Bringing the bounds together provides the PSPACE upper bound. □

Theorem 4.12. *Functional correctness for PAS is PSPACE-hard.*

Proof Sketch. We simulate deterministic Turing machines running in PSPACE. The challenge to be addressed is that within Turing computations we are dealing with a stable device for permanent storage of the information we need. As for our automata, the situation is the opposite, namely, each time, reading m stored in channel $R_{A,B}(m)$ nullifies $R_{A,B}$. The full proof is given in the Appendix. □

For a k-bounded LAS, it makes sense to consider a parameterised version of FCP in which the bound k is considered an additional part of the input to the decision problem.

Theorem 4.13. *Functional correctness for k-bounded LAS is in coNP, where k is considered an additional part of the input.*

Proof Sketch. Let $\mathcal{N} = (\mathcal{A}, \mathcal{R})$ be a k-bounded LAS. The number of actions in a hyper-cycle is polynomial in the size of $\mathcal{N}, k$. Therefore, the existence of an appropriate sequence of actions leading to a critical configuration can be guessed in NP and hence, FCP can be done in coNP. □

Theorem 4.14. *FCP for LAS is coNP-hard.*

Proof Sketch. We simulate 3-SAT problems by 1-bounded LAS. The full proof is omitted because of space constraints.

5 INTRUDER MODEL

This Section introduces an intruder model for I4.0, based on the Dolev-Yao intruder model (Dolev and Yao, 1983), but adapted to I4.0 applications following the findings of the BSI security assessment (Fiat et.al., 2017) of OP-CUA. The assessment concludes that the major threats arise from the injection and tampering of messages. Our model also supports that the intruder blocks messages.

Since messages communicated in channels of automata systems are not encrypted, we assume that

the intruder is familiar with all the signal constants that can be exchanged between automata in the system. Hence, differently from the DY intruder models such as the ones in (Durgin et al., 2004; Kanovich et al., 2014; Urquiza et al., 2019) that include *e.g.*, intruder rules for pairing, encryption and decryption of messages, such intruder rules do not contribute to the power of intruder here. In other words, the intruder does not have to eavesdrop in order to collect some knowledge about the system. Instead, intruder already knows all the possible messages that can be exchanged in an AS.

Formally, intruders are modelled as finite automata that control the network, that is, have access to all channels.

Definition 5.1 (Intruder)**.** *An* intruder *I is represented as a one state automaton defined by a finite set of rules $\mathcal{R}_I$ of the form:*

$$R_{A,C}(*) \longrightarrow R_{A,C}(\mathtt{m}) \quad (2)$$
$$R_{A,C}(\mathtt{m}) \longrightarrow R_{A,C}(*) \quad (3)$$
$$R_{A,C}(\mathtt{m}) \longrightarrow R_{A,C}(\mathtt{m}') \quad (4)$$

where $R_{A,C}$ is any channel and $\mathtt{m}$ and $\mathtt{m}'$ are any message symbols of any AS, such that $\mathtt{m},\mathtt{m}' \neq *$.

Remark 5.2. *Since the automata representing intruders have only one state, for simplicity, we abbreviate the form of Eq.(1) by omitting the facts denoting automata states in rules Eq.(2), Eq.(3) or Eq.(4).*

Using the rule Eq.(2) an intruder is capable of injecting a signal $\mathtt{m}$ into an empty channel. Using the rule Eq.(3) an intruder removes a signal $\mathtt{m}$ from a channel, while by using the rule Eq.(4) an intruder modifies a signal $\mathtt{m}$ into a signal $\mathtt{m}'$.

By restricting the type of rules and/or imposing some other restrictions on the intruder rules, we can consider intruders of various capabilities, *e.g.*, intruders that can only read/remove sent messages, obstructing communication between automata in the system. By additionally bounding the number of intruder interventions, we introduce the notion of bounded intruders. For our complexity results, we, in particular, consider a bounded intruder that is allowed to interfere with the system only once, *i.e.*, we search for attack traces with a single intruder action.

Definition 5.3 (Bounded Intruder)**.** *A* bounded intruder *I is an intruder that when interfering with some AS $\mathcal{N} = (\mathcal{A},\mathcal{R})$ is allowed to use only a bounded number of actions of type Eq.(2), Eq.(3) or Eq.(4), on any channel $R_{A,C} \in \mathcal{R}$ using some signal(s) $\mathtt{m},\mathtt{m}' \in M_A$, $A \in \mathcal{A}$.*

There are several motivations for considering bounded intruders. The first one comes from I4.0 applications themselves. As discussed in (Lanotte et al., 2020), I4.0 are Cyber-Physical systems where each action takes time, including intruder actions. This means that the intruder cannot send an unbounded number of actions during an application period. This is similar to notions of progressing systems (Kanovich et al., 2013). A second analogy/motivation is that bounded intruder models correspond to bounded verification problems, such as in bounded model checking (Biere et al., 2003).

5.1 Example Attack by Message Insertion on PnP AS

An example attack on the automata system PnP by the intruder defined above is described below. Recall from Section 4.1 that a configuration denoting that the vacuum is switched off while the arm is moving left, specified by the facts $Q_{\mathsf{vac}}(\mathsf{off}), Q_{\mathtt{track}}(\mathsf{mvL}), Q_{\mathsf{ctl}}(\mathsf{ROn})$ is critical.

Any intruder having the capability to insert messages in the channel $R_{\mathsf{ctl,vac}}$ is capable of performing the attack. Starting from the initial configuration:

$$Q_{\mathsf{ctl}}(\mathsf{Init}), Q_{\mathsf{track}}(\mathsf{L}), Q_{\mathsf{vac}}(\mathsf{off}), R_{\mathsf{ctl,vac}}(*), R_{\mathsf{ctl,track}}(*), R_{\mathsf{track,ctl}}(*), R_{\mathsf{vac,ctl}}(*), R_{\mathsf{ctl,ctl}}(*)$$

consecutive application of the following system rules $\mathtt{c}_1, \mathtt{c}_2, \mathtt{t}_1, \mathtt{t}_2, \mathtt{c}_3, \mathtt{v}_1, \mathtt{c}_4, \mathtt{t}_3$ leads to configuration:

$$Q_{\mathsf{ctl}}(\mathsf{ROn}), Q_{\mathsf{track}}(\mathsf{mvL}), Q_{\mathsf{vac}}(\mathsf{on}), R_{\mathsf{ctl,vac}}(*), R_{\mathsf{ctl,track}}(*), R_{\mathsf{track,ctl}}(*), R_{\mathsf{vac,ctl}}(*), R_{\mathsf{ctl,ctl}}(*).$$

An intruder then inserts a signal into the channel from ctl to vac using the intruder rule of type Eq (2): $R_{\mathsf{ctl,vac}}(*) \longrightarrow R_{\mathsf{ctl,vac}}(\mathsf{VacOff})$, obtaining:

$$Q_{\mathsf{ctl}}(\mathsf{ROn}), Q_{\mathsf{track}}(\mathsf{mvL}), Q_{\mathsf{vac}}(\mathsf{on}), R_{\mathsf{ctl,vac}}(\mathsf{VacOff}), R_{\mathsf{ctl,track}}(*), R_{\mathsf{track,ctl}}(*), R_{\mathsf{vac,ctl}}(*), R_{\mathsf{ctl,ctl}}(*).$$

Application of rule $\mathtt{v}_2$ leads to critical configuration:

$$Q_{\mathsf{ctl}}(\mathsf{ROn}), Q_{\mathsf{track}}(\mathsf{mvL}), Q_{\mathsf{vac}}(\mathsf{off}), R_{\mathsf{ctl,vac}}(*), R_{\mathsf{ctl,track}}(*), R_{\mathsf{track,ctl}}(*), R_{\mathsf{vac,ctl}}(\mathsf{NoVac}), R_{\mathsf{ctl,ctl}}(*).$$

5.2 Example Attack on a LAS: Breaking a Hyper-cycle

Consider the following example of an AS that is a functionally correct LAS. Let $\mathcal{N} = (\mathcal{A},\mathcal{R})$, where $\mathcal{A} = \{A_1, A_2, A_3\}$, $A_1 = (S_1, q_0^1, M, X_1)$, $A_2 = (S_2, q_0^2, M, X_2)$, $A_3 = (S_3, q_0^3, M, X_3)$, $S_1 = \{q_0^1, q_1^1, q_2^1, q_3^1\}$, $S_2 = \{q_0^2, q_1^2, q_2^2\}$, $S_3 = \{q_0^3, q_1^3\}$ and $\mathcal{R} = \{R_{A_1,A_2}, R_{A_2,A_1}, R_{A_2,A_2}, R_{A_1,A_3}, R_{A_3,A_1}\}$. Let $M = \{*, \mathtt{a}, \mathtt{b}, \mathtt{c}\}$ be the set of signals of all automata, and the set of instructions X_i of each automaton A_i defined as per Figure 3. Let critical configurations of $\mathcal{N}$ be those that contain the fact $Q_{A_1}(q_2^1)$.

The only hyper-cycle of $\mathcal{N}$ consists of the consecutive application of rules s_1, s_2, r_1, r_2, s_3, given in

$$
\begin{array}{lll}
X_1: & r_1: & Q_{A_1}(q_0^1), R_{A_2,A_1}(\mathtt{b}) \longrightarrow Q_A(q_1^1), R_{A_2,A_1}(*) \\
& r_2: & Q_{A_1}(q_1^1), R_{A_1,A_2}(*) \longrightarrow Q_A(q_0^1), R_{A_1,A_2}(\mathtt{a}) \\
& r_3: & Q_{A_1}(q_1^1), R_{A_2,A_1}(\mathtt{c}) \longrightarrow Q_{A_1}(q_2^1), R_{A_2,A_1}(*) \\
& r_4: & Q_{A_1}(q_2^1), R_{A_1,A_3}(*), \longrightarrow Q_{A_1}(q_0^1), R_{A_1,A_3}(\mathtt{a}) \\
& r_5: & Q_{A_1}(q_0^1), R_{A_3,A_1}(\mathtt{b}) \longrightarrow Q_A(q_3^1), R_{A_3,A_1}(*) \\
& r_6: & Q_{A_1}(q_3^1), R_{A_1,A_3}(*) \longrightarrow Q_A(q_0^1), R_{A_1,A_3}(\mathtt{a}) \\
X_2: & s_1: & Q_{A_2}(q_0^2), R_{A_2,A_2}(*) \longrightarrow Q_{A_2}(q_1^2), R_{A_2,A_2}(b) \\
& s_2: & Q_{A_2}(q_1^2), R_{A_2,A_1}(*), R_{A_2,A_2}(b) \longrightarrow \\
& & \quad Q_{A_2}(q_2^2), R_{A_2,A_1}(\mathtt{b}), R_{A_2,A_2}(*) \\
& s_3: & Q_{A_2}(q_2^2), R_{A_1,A_2}(\mathtt{a}) \longrightarrow Q_A(q_0^2), R_{A_1,A_2}(*) \\
X_3: & p_1: & Q_{A_3}(q_0^3), R_{A_1,A_3}(\mathtt{a}), R_{A_3,A_1}(*) \longrightarrow \\
& & \quad Q_{A_3}(q_1^3), R_{A_1,A_3}(*), R_{A_3,A_1}(\mathtt{b}) \\
& p_2: & Q_{A_3}(q_1^3), R_{A_1,A_3}(*), \longrightarrow Q_{A_3}(q_0^3), R_{A_1,A_3}(*)
\end{array}
$$

Figure 3: Instructions of the Example LAS.

Figure 3. Notice that the hypercycle contains no rules of A_3. Starting from the initial configuration $\mathcal{S}_0$, any infinite trace of $\mathcal{N}$ is obtained as the concatenation of this hyper-cycle. Hence, $\mathcal{N}$ is periodic.

Moreover, each automaton rule is applied at most once in a hyper-cycle, hence $\mathcal{N}$ is an LAS. $\mathcal{N}$ is functionally correct since the fact $Q_{A_1}(q_2^1)$ is not reachable from $\mathcal{S}_0$ using only system rules.

However, in the presence of an intruder, critical configuration is reachable. There is an attack on system $\mathcal{N}$ by message insertion by an intruder, using only the rule $i_c: R_{A_2,A_1}(*) \longrightarrow R_{A_2,A_1}(\mathtt{c})$ **once**. A trace from $\mathcal{S}_0$ starting with rules s_1, s_2, r_1, followed by the intruder rule i_c reaches the configuration to which the rule r_3 can be applied:

$$
\begin{array}{l}
Q_{A_1}(q_0^1), Q_{A_2}(q_0^2), Q_{A_3}(q_0^3), R_{A_2,A_2}(*), \\
\quad R_{A_1,A_2}(*), R_{A_2,A_1}(*), R_{A_1,A_3}(*), R_{A_3,A_1}(*) \longrightarrow_{s_1} \\
Q_{A_1}(q_0^1), Q_{A_2}(q_1^2), Q_{A_3}(q_0^3), R_{A_2,A_2}(\mathtt{b}), \\
\quad R_{A_1,A_2}(*), R_{A_2,A_1}(*), R_{A_1,A_3}(*), R_{A_3,A_1}(*) \longrightarrow_{s_2} \\
Q_{A_1}(q_0^1), Q_{A_2}(q_2^2), Q_{A_3}(q_0^3), R_{A_2,A_2}(*), \\
\quad R_{A_1,A_2}(*), R_{A_2,A_1}(\mathtt{b}), R_{A_1,A_3}(*), R_{A_3,A_1}(*) \longrightarrow_{r_1} \\
Q_{A_1}(q_1^1), Q_{A_2}(q_2^2), Q_{A_3}(q_0^3), R_{A_2,A_2}(*), \\
\quad R_{A_1,A_2}(*), R_{A_2,A_1}(*), R_{A_1,A_3}(*), R_{A_3,A_1}(*) \longrightarrow_{i_c} \\
Q_{A_1}(q_1^1), Q_{A_2}(q_2^2), Q_{A_3}(q_0^3), R_{A_2,A_2}(*), \\
\quad R_{A_1,A_2}(*), R_{A_2,A_1}(\mathtt{c}), R_{A_1,A_3}(*), R_{A_3,A_1}(*) \longrightarrow_{r_3} \\
Q_{A_1}(q_2^1), Q_{A_2}(q_2^2), Q_{A_3}(q_0^3), R_{A_2,A_2}(*), \\
\quad R_{A_1,A_2}(*), R_{A_2,A_1}(*), R_{A_1,A_3}(*), R_{A_3,A_1}(*)
\end{array}
$$

By inserting the signal c into the appropriate channel, intruder causes A_1 not to procede within the hyper-cycle, but instead to apply rule r_3. The resulting configuration is critical as it contains the fact $Q_{A_1}(q_2^1)$. Therefore, the above trace represents an attack. Furthermore, the above finite attack trace can be extended into an infinite trace with no hyper-cycles of $\mathcal{N}$, in which automata A_1 and A_3 play an infinite ping pong game, while A_2 is stuck. There are no hyper-cycles in the above attack trace. It suffices for an intruder to apply just a single action of message insertion to perform the attack and change the behavior of the system.

This example shows that even a well designed, functionally correct PAS, including the case of a LAS, in the presence of an intruder is no longer periodic.

5.3 Security Complexity Results

We investigate the complexity of deciding whether a functionally correct AS can reach a critical configuration in the presence of an intruder.

Definition 5.4 (Security Problem for Functionally Correct Systems (SP-FCS))**.** *Given an AS $\mathcal{N}$ that is functionally correct and an intruder model I, is there a trace using the rules of $\mathcal{N}$ and I leading from the initial configuration to a critical configuration of $\mathcal{N}$?*

We investigate the complexity of the SP-FCS for different classes of AS and intruders. Recall from Section 5.2 that, given an AS that is a LAS or a PAS, once the intruder is present it may no longer be either.

Theorem 5.5. *The SP-FCS is PSPACE-complete. The SP-FCS PSPACE-complete even in the case the intruder in question can apply only one action. These problems are still PSPACE-complete even in the case of a PAS and a LAS, and even in the case the intruder can apply only one action.*

Proof Sketch. For the lower bound, in order to incorporate the intruder, we modify the proof of Theorem 4.12 accordingly. Upper bound follows from Theorem 4.11. For more details see Appendix.

The above results are summarized in Table 1.

6 AUTOMATED VERIFICATION

The formal models, verification problems, and complexity results support the automated security verification of I4.0 applications. We demonstrate this by carrying out a number of experiments based on the Maude formalization described in (Nigam and Talcott, 2019). Our experiments are based on the following variations of the example described in Section 3.

PnP - This is the scenario described in Section 3.

2PnP - This scenario is a LAS containing two instances of PnP and a coordinator that ensures the start of each instance of PnP starts at the same time, namely, the start of the hyper-cycle.

PnP-2Msgs - This scenario modifies the logic of the PnP so that the track at the right (where the caps are) waits for two signals to head leftwards (where the cap has to be placed): GoL from ctl and HasVac/NoVac from vac; and when vac is on it requires two signals to turn off: VacOff from ctl and

Table 2: Model-checking results for the SP-FCS using Maude for different scenarios. The values in parentheses, $\times n$, for a scenario and bound on the intruder, denotes that Maude traversed n times more configurations than the scenario PnP with the same value for the bound on the intruder. The experiments were run on a MacBook Pro, 2.4 Ghz Intel Core i5, 16GB memory.

Scenario	Bound on Intruder	Number of Configurations Explored	Time(ms)	SP-FCS
PnP	0	23	4	no
	1	84	11	yes
	2	406	47	yes
	3	1651	178	yes
2PnP	0	84 ($\times$3.7)	40	no
	1	388 ($\times$4.6)	182	yes
	2	2873 ($\times$7.1)	1409	yes
	3	26440 ($\times$16.0)	19631	yes
PnP-2Msgs	0	29 ($\times$1.3)	40	no
	1	722 ($\times$8.5)	177	no
	2	1854 ($\times$4.6)	912	yes
	3	10248 ($\times$6.2)	4965	yes
2PnP-2Msgs	0	114 ($\times$4.9)	88	no
	1	6814 ($\times$81.1)	5277	no
	2	22179 ($\times$54.1)	18208	yes
	3	153824 ($\times$93.1)	225898	yes

atL from track. Intuitively, this means that the intruder would need at least two actions to lead this system to a critical configuration.

2PnP-2Msgs - This scenario is similar to the scenario 2PnP, but uses PnP-2Msgs instead of PnP.

For each scenario, we carried out experiments in Maude to check the reachability of critical configurations in the presence of a bounded intruder with the bound on the number of intrusions between 0 and 3. Note that unreachability with the bound 0 corresponds to checking whether the system is functionally correct. For the scenarios, we use the critical configurations as described in Section 3. Table 2 summarizes experiments on the four scenarios described above.

These experiments show that it is feasible in practice to formally verify simple scenarios and even more complicated ones. However, as expected from our complexity results, the computational effort, *i.e.*, the number of configurations explored increases exponentially as we increase the size of the system. Moreover, increasing the bound on intruders impacts the search space. Higher bound values mean that intruders are capable to carry out more complex attacks, *e.g.*, in the scenarios 2PnP and 2PnP-2Msgs the intruder needs at least two actions to carry out an attack.

7 CONCLUSIONS AND FUTURE WORK

This paper introduces a formal model based on MSR for representation and verification of automated interconnected systems such as I4.0 applications. Within the general framework, several classes of systems are identified, each with specific properties that are interesting, and motivated by concrete applications. Different verification problems are investigated and a comprehensive collection of complexity results is obtained, including several security complexity results involving different types of intruders.

There is a number of ways the model of automata systems introduced in this paper can be extended. For example, systems with smart devices as components, systems of distributed manufacturing, and similar systems could be considered, for which the communication among systems components would follow cryptographic networking protocols. At the same time, an intruder with encryption capabilities, more similar to DY intruder could be modelled.

In order to avoid some false positives, the model could be extended with time, using timed MSR models (Kanovich et al., 2017) obtaining timed system and intruder models that take into account physical properties such as distances and processing time.

Similar to the work in (AlTurki et al., 2018), statistical model-checking could be applied to investigate the success rates of various intruder strategies.

Additionally, systems and intruders with various resource-sensitive features may lead to investigations of other verification problems, such as the ones considered in (Urquiza et al., 2019).

For each of these extensions, we plan to investigate abstraction techniques and properties similar to (Nigam and Talcott, 2020) that relate various extensions of our model.

ACKNOWLEDGMENTS

Part of this work was done during the visits to the University of Pennsylvania by Alturki, Ban Kirigin, Kanovich, Nigam, and Talcott, which were partially supported by ONR grant N00014-15-1-2047 and by the University of Pennsylvania. Ban Kirigin is supported in part by the Croatian Science Foundation under the project UIP-05-2017-9219. The work of Max Kanovich was partially supported by EPSRC Programme Grant EP/R006865/1: "Interface Reasoning for Interacting Systems (IRIS)." Nigam is partially supported by NRL grant N0017317-1-G002, and CNPq grant 303909/2018-8. Scedrov is partially supported by ONR grants N00014-20-1-2635 and N00014-18-1-2618. Talcott was partially supported by ONR grants N00014-15-1-2202 and N00014-20-1-2644, and NRL grant N0017317-1-G002.

REFERENCES

Ademaj et al. (2019). Time sensitive networks for flexible manufacturing testbed - description of converged traffic types, IIC white paper.

AlTurki, M. A., Kanovich, M., Ban Kirigin, T., Nigam, V., Scedrov, A., and Talcott, C. (2018). Statistical model checking of distance fraud attacks on the Hancke-Kuhn family of protocols. In *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and PrivaCy*, pages 60–71. ACM.

Biere, A., Cimatti, A., Clarke, E. M., Strichman, O., and Zhu, Y. (2003). Bounded model checking. *Advances in Computers*, 58:117–148.

Cyberattack Has Caused Confirmed Physical Damage for the Second Time Ever. (2015). Available at https://www.wired.com/2015/01/german-steel-mill-hack-destruction/.

Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., and Talcott, C. (2007). *All About Maude: A High-Performance Logical Framework*, volume 4350 of *LNCS*. Springer.

Dolev, D. and Yao, A. (1983). On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208.

Durgin, N. A., Lincoln, P., Mitchell, J. C., and Scedrov, A. (2004). Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 12(2):247–311.

ENISA (2018). Good practices for security of internet of things in the context of smart manufacturing.

Fiat, M. and et.al. (2017). OPC UA security analysis.

Kanovich, M., Ban Kirigin, T., Nigam, V., and Scedrov, A. (2013). Bounded memory protocols and progressing collaborative systems. In *ESORICS*, pages 309–326.

Kanovich, M. I., Ban Kirigin, T., Nigam, V., and Scedrov, A. (2014). Bounded memory Dolev-Yao adversaries in collaborative systems. *Inf. Comput.*, 238:233–261.

Kanovich, M. I., Ban Kirigin, T., Nigam, V., Scedrov, A., and Talcott, C. L. (2017). Time, computational complexity, and probability in the analysis of distance-bounding protocols. *Journal of Computer Security*, 25(6):585–630.

Kanovich, M. I., Rowe, P., and Scedrov, A. (2011). Collaborative planning with confidentiality. *Journal of Automated Reasoning*, 46(3-4):389–421.

Lanotte, R., Merro, M., Munteanu, A., and Viganò, L. (2020). A formal approach to physics-based attacks in cyber-physical systems. *ACM Trans. Priv. Secur.*, 23(1).

Nigam, V. and Talcott, C. (2019). Formal security verification of industry 4.0 applications. In *ETFA, Special Track on Cybersecurity in Industrial Control Systems*.

Nigam, V. and Talcott, C. (2020). Automated construction of security integrity wrappers for industry 4.0 applications. In *International Workshop on Rewriting Logic and its Applications (WRLA)*.

Savage, J.E.(1998). Models of computation. Addison-Wesley Reading, MA

Urquiza, A. A., AlTurki, M. A., Kanovich, M., Ban Kirigin, T., Nigam, V., Scedrov, A., and Talcott, C. (2019). Resource-bounded intruders in denial of service attacks. In *32nd Computer Security Foundations Symposium (CSF)*, pages 382–396. IEEE.

Yoong, L. H., Roop, P. S., Bhatti, Z. E., and Kupz, M. M. Y. (2015). *Model-Driven Design Using IEC 61499: A Synchronous Approach for Embedded Automation Systems*. Springer.

Zoitl, A. and Lewis, R. (2014). *Modelling control systems using IEC 61499*. Control Engineering Series 95. The Institution of Electrical Engineers, London.

APPENDIX

PSPACE-hardness in Theorem 4.12

Theorem 4.12. *FCP for PAS is PSPACE-hard.*

Remark 7.1. *For the sake of readability, here, and henceforth, we will abbreviate Eq.(1) as:*

$$\mathtt{q}, R_{B_1,A}(\mathtt{m}_1), .., R_{B_k,A}(\mathtt{m}_k) \to \mathtt{q}', R_{A,C_1}(\mathtt{m}'_1), .., R_{A,C_\ell}(\mathtt{m}'_\ell) \quad (5)$$

The PSPACE decision problem can be defined as:

"*Given a Turing machine M running in space m, determine whether there is a binary string x of length m so that x is accepted by M.*"

We reformulate the problem in terms of $\widetilde{M}$, which deals only with one and the same initial configuration fixed in advance.

Lemma 7.2. *Given a deterministic Turing machine M running, say, in space $m = n/3$, we construct a deterministic Turing machine $\widetilde{M}$ running in space n so that for its fixed initial tape of the form $\overbrace{aa..a}^{n\ times}$ and its initial*

state q_1, $\widetilde{M}$ **always terminates** *but in one of the two states: $\widetilde{q_0}$ or $\widetilde{q_1}$.*

Moreover, $\widetilde{M}$ terminates in $\widetilde{q_0}$ iff one can find a binary string x of length m so that x is accepted by M.

Besides, $\widetilde{M}$ is constructed so that $\widetilde{M}$ starts with its initial state q_1 at the leftmost position on the tape and terminates with $\widetilde{q_0}$ or with $\widetilde{q_1}$ at the same leftmost position on the tape. There are no moves in $\widetilde{M}$ from $\widetilde{q_0}$ or $\widetilde{q_1}$. Each $\widetilde{M}$'s command, $q\xi \to q'\eta D$, must "move" to the left, which is marked by $D = -1$, or to the right, which is marked by $D = +1$. □

Our goal is to mimic the terminated computation performed by $\widetilde{M}$ in terms of hyper-cycles from A_0 to A_0, where the automaton A_0, the 'main controller' in our system, is specified by the following instructions (r_0 is its initial state)

$$\begin{cases} r_0 & \longrightarrow & r'_0, R_{A_0,A_1}(p) \\ R_{B_{n+1},A_0}(\widetilde{q}), r'_0, & \longrightarrow & r_0, \quad \text{where } \widetilde{q} \in \{\widetilde{q_0}, \widetilde{q_1}\} \end{cases} \tag{6}$$

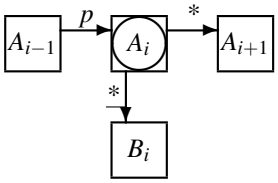
Initially all channels are empty. A_0 starts its hyper-cycle with sending signal p to A_1 via channel R_{A_0,A_1}. Then A_0 is waiting for a signal $\widetilde{q}$ sent from B_{n+1} to end its hyper-cycle.

We develop our AS by designing the automata we need step by step using a chain of lemmas. *To ease technicalities, we define the automata at hand only in terms of the tasks the automata should perform.*

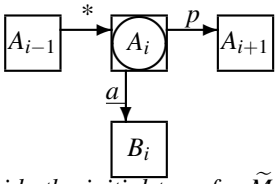
As signals, we use q, ξ, η, and q', etc., the tape symbols and states of $\widetilde{M}$. We use p as a specific extra signal. In addition, we introduce a polynomial number of fresh signals, $\langle q', \eta, D\rangle$, to represent triples of the form (q', η, D).

Providing $\widetilde{M}$'s Initial Tape

Lemma 7.3. *For $1 \le i \le n$, we design A_i so that A_i can transform a precondition of the form*



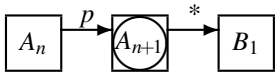
into a postcondition of the form



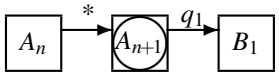
Then we provide the initial tape for $\widetilde{M}$, aa..a, by sequential execution of automata A_1, A_2, …, A_n, resulting in the 'initial' non-empty channels $R_{A_1,B_1}(a)$, $R_{A_2,B_2}(a)$, …, $R_{A_n,B_n}(a)$.

Simulating $\widetilde{M}$'s Computations

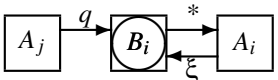
Lemma 7.4. *To provide the correct start of $\widetilde{M}$ with its initial state q_1, we design A_{n+1} so that A_{n+1} transforms the precondition produced by the n-th step of Lemma 7.3*



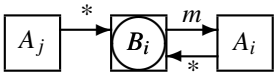
into a postcondition of the form



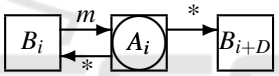
Lemma 7.5. *Given a Turing command $q\xi \to q'\eta D$, first we design B_i, $i = 1, .., n$, so that B_i can transform a precondition of the form, $j \ne i$,*



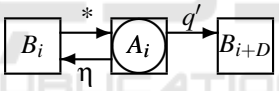
into the following postcondition, where $m = \langle q', \eta, D\rangle$:



and we modify A_i so that, in addition to Lemma 7.3, A_i can transform a precondition of the form (recall $D = \pm 1$)



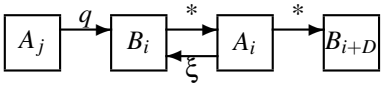
into the following postcondition,



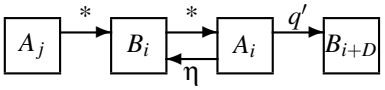
Lemma 7.6. *Any computation performed by $\widetilde{M}$ can be one-to-one simulated by running sequentially the corresponding ordered pairs of automata B_i and A_i.*

Proof. Suppose that, being in state q and scanning ξ in i-th tape cell, $\widetilde{M}$ applies its command $q\xi \to q'\eta D$.

By induction we represent the enabling conditions for the above $\widetilde{M}$'s move as a reachable configuration of the form



By Lemma 7.5 the following configuration that represents the enabling conditions for the next $\widetilde{M}$'s move, is reachable:



Lemma 7.7. *Our system behaves deterministically.*

Proof. By induction we show that the enabling conditions are not overlapped at any moment, so that no more than one automaton instruction can be applied at the current moment. □

Lemma 7.8. *For $\widetilde{q} \in \{\widetilde{q_0}, \widetilde{q_1}\}$, $\widetilde{M}$ terminates in $\widetilde{q}$ iff $R_{A_2,B_1}(\widetilde{q})$ is reachable within our system.*
In particular, $R_{A_2,B_1}(\widetilde{q_0})$ is reachable iff one can find a binary string x of length m so that x is accepted by M.

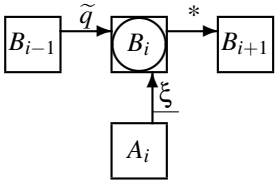
Proof. The direction "only if" is the most problematic. Suppose that $R_{A_2,B_1}(\widetilde{q})$ is reachable, but $\widetilde{M}$ terminates in some $\widetilde{q'}$. Then by Lemma 7.6 $R_{A_2,B_1}(\widetilde{q'})$ must be reachable as well, and Lemma 7.7 requires $\widetilde{q'} = \widetilde{q}$. □

Corollary 7.9. *Claiming $R_{A_2,B_1}(\widetilde{q_0})$ critical provides PSPACE-hardness for functional correctness.*

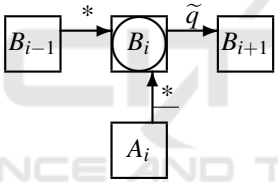
Proof. Follows from Lemmas 7.2 and 7.8. □

Collecting Garbage

Lemma 7.10. *For $1 \leq i \leq n$, we modify B_i so that, in addition to Lemma 7.5, B_i can transform the precondition*



into the 'cleaner' postcondition



For $i = 1$, we take A_2 as B_{i-1}. □

At the end of the hyper-cycle, we nullify all channels R_{A_i,B_i} with Lemma 7.10 applied sequentially.

Our system given in this Section is a periodic system with a unique hyper-cycle from A_0 to A_0. □

PSPACE-hardness in Theorem 5.5

Theorem 5.5: *The SP-FCS is PSPACE-complete.*
The SP-FCS PSPACE-complete even in the case the intruder in question can apply only one action.
These problems are still PSPACE-complete even in the case of a PAS and a LAS, and even in the case the intruder can apply only one action.

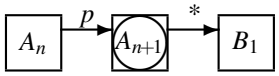
We investigate a restricted decision problem:

"*Given as input to the problem: a locally bounded periodic AS, which is functionally corect, and an intruder, which can apply only one action, determine whether a critical configuration is reachable within the system enriched with the intruder action.*"

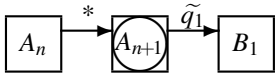
As input to the problem we take an AS from Definition 7.11 and an intruder from Definition 7.14.

Definition 7.11. *For a fixed $\widetilde{M}$ from Lemma 7.2, we take the system introduced in this Section and replace only one lemma, Lemma 7.4, with Lemma 7.12.*

Lemma 7.12. *We update A_{n+1} so that A_{n+1} can transform the precondition produced by the n-th step of Lemma 7.3*



into a postcondition of the form



As critical we take system configurations which contain $R_{A_2,B_1}(\widetilde{q_0})$ at some moment of execution.

Lemma 7.13. *The system given in Definition 7.11 is a LAS which is functionally correct.*

Proof. According to Lemmas 7.3, 7.12, and 7.10, we can develop a unique hyper-cycle from A_0 to A_0 by running sequentially the following automata

$$A_0, A_1, A_2, \ldots, A_n, A_{n+1}, B_1, B_2, \ldots, B_n, B_{n+1}, A_0.$$

Initially all channels are empty. A_0 starts a hyper-cycle with sending signal p to A_1 via channel R_{A_0,A_1}. Sequentially running $A_1, \ldots, A_n$ results in the non-empty channels $R_{A_1,B_1}(a)$, ..., $R_{A_n,B_n}(a)$. At once Lemma 7.12 redirects the execution to the garbage collecting Lemma 7.10, which makes channels empty. Consuming $R_{B_{n+1},A_0}(\widetilde{q_1})$ at state r'_0, A_0 ends the current hyper-cycle.

Notice that the automaton instructions involved in the above execution in question have been applied no more than once. □

Definition 7.14. *Let an intruder be able to attack the updated A_{n+1}, by changing its outgoing signal $\widetilde{q_1}$ into the signal q_1 by means of the following action that modifies the channel R_{A_{n+1},B_1}:*

$$R_{A_{n+1},B_1}(\widetilde{q_1}) \longrightarrow R_{A_{n+1},B_1}(q_1) \qquad (7)$$

Lemma 7.15. *For $\widetilde{q} \in \{\widetilde{q_0}, \widetilde{q_1}\}$, $\widetilde{M}$ terminates in $\widetilde{q}$ iff $R_{A_2,B_1}(\widetilde{q})$ is reachable within our system in Definition 7.11 enriched with the intruder action (7).*

Proof. Similar to Lemma 7.8.

At the moment when A_{n+1} provides $R_{A_{n+1},B_1}(\widetilde{q_1})$ by Lemma 7.12, the intruder redirects the execution to 'sleeping' automata by modifying a channel of the form $R_{A_{n+1},B_1}(\widetilde{q_1})$ into $R_{A_{n+1},B_1}(q_1)$.

The result is that at the next moment B_1 starts not with $\widetilde{q_1}$ but with q_1, *the true initial state of $\widetilde{M}$.* □

Bringing all things together we complete the proof of Theorem 5.5. □