SN Computer Science (2022) 3:356
https://doi.org/10.1007/s42979-022-01223-9

ORIGINAL RESEARCH

On the Security and Complexity of Periodic Systems

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

Received: 30 September 2021 / Accepted: 21 May 2022

Abstract
Recent years have seen a tremendous increase in the reliance of industrial systems on a variety of interconnected components ranging in complexity from simple sensors to more complex cyber-physical and Internet of Things (IoT) devices, a class of systems that is often referred to as Industry 4.0 (I4.0). Increased connectivity and the proliferation of insecure components present an opportunity for cyber attacks that could in practice inflect far-reaching damage. We present in this paper a formal modeling and analysis approach of I4.0 applications and their safety and security properties. We introduce formal models of I4.0 applications as automata systems (AS) expressed as theories in Multiset Rewriting (MSR). We also identify different subclasses of AS, reflecting different types of I4.0 requirements, such as periodicity. Furthermore, we model different levels of threats to the system by proposing a range of intruder models based on the number of actions that intruders can use. These models are used to investigate the complexity of two types of problems: functional correctness (safety) and vulnerability to attacks (security). Finally, we demonstrate that periodic systems are amenable to automated verification by describing an executable specification of these models using the rewriting tool Maude and carrying out various experiments.

Keywords Formal methods · Verification · Security · Multiset rewriting · Industry 4.0 · Complexity

Introduction

Recent years have seen a tremendous increase in the reliance of industrial systems on a wide range of networked devices, ranging from simple sensors and controllers all the way to cyber-physical and Internet of Things (IoT) devices. This trend, referred to as Industry 4.0 (I4.0), has been primarily driven by the need to improve production efficiency and enable process agility and product personalization. However, the combination of flexible interconnectivity and insecure devices used in practice has created new opportunities for cyber attacks, which in an industrial setting, can potentially result in human suffering or material damage. A notorious example of such attacks is the targeted attack on a steel mill in a steel plant that caused massive damage and forced the plant to stop production, resulting in significant financial loss [5].

To enable analyzing I4.0 applications and increase their safety and security, the IEC 61499 international standard for

This article is part of the topical collection "Information Systems Security and Privacy" guest edited by Steven Furnell and Paolo Mori.

✉ Musab A. Alturki
musab.alturki@runtimeverification.com

✉ Tajana Ban Kirigin
bank@math.uniri.hr

Max Kanovich
m.kanovich@ucl.ac.uk

Vivek Nigam
vivek@ci.ufpb.br

Andre Scedrov
scedrov@math.upenn.edu

Carolyn Talcott
clt@csl.sri.com

1 KFUPM, Dhahran, Saudi Arabia
2 Runtime Verification Inc., Urbana, IL, USA
3 Faculty of Mathematics, University of Rijeka, Rijeka, Croatia
4 University College London, London, UK
5 Federal University of Paraíba, João Pessoa, Brazil
6 Munich Research Center, Huawei, Munich, Germany
7 University of Pennsylvania, Philadelphia, PA, USA
8 SRI International, Menlo Park, CA, USA

Published online: 02 July 2022

distributed industrial control systems [24, 25] specifies the possible behaviours of the entire industrial application or process using a platform-independent, event-driven application model. The model is fundamentally composed of elements, called *function blocks* (FBs), that interact through data and event interfaces [24, 25]. An FB can range in complexity from a simple state-transition system to a more complex network of FBs (a composite FB). This model can also capture periodically executing applications, which are commonly used in industrial settings.

A number of attempts to systematically model I4.0 systems and analyze their security properties have appeared in the literature, but they were mostly informal. They include the relatively recent Federal Office for Information Security (BSI) report on the security of OPC-UA (machine to machine communication protocol for industrial automation) [11] and the study on good practices for IoT security in the context of I4.0 by the European Union Agency for Cybersecurity (ENISA) [10].

We present in this paper a formal modeling and analysis approach of I4.0 applications and their safety and security properties. The underlying formal framework is based on Multiset Rewriting (MSR) [8]. Motivated by the requirements of I4.0 applications, we propose three levels of refinements of MSR models that model I4.0 systems:

Automata Systems (AS) are used to represent systems similar to those specified in the IEC 61499 standard. In particular, FBs are specified as possibly non-deterministic Mealy machines [21] that interact by performing local computations and exchanging events.
Periodic Automata Systems (PAS) refine AS by incorporating the assumption that I4.0 applications are periodic. Such an I4.0 application performs a collection of tasks by executing its FBs periodically, i.e., repeating systems cycles.
Locally Bounded Periodic Automata Systems (LAS) further refine PAS by bounding the number of executions of each FB within one system cycle.

We first consider the safety properties of I4.0 applications, formalized as properties of *functional correctness*. The executions of functionally correct systems are guaranteed to represent the correct execution of the work process within the closed system, without interference from outside. We investigate the complexity of the *Functional Correctness Problem* (FCP), which consists in verifying that a system does not exhibit a behaviour that leads to a bad configuration representing an unsafe situation that may be threatening to humans or may cause financial loss. FCP can be seen as checking that the system behaves correctly in the absence of an intruder.

We then study the security of I4.0 applications, i.e., the behaviour of the systems in the presence of intruders. When considering intruder models, we follow the findings of the BSI report on OPC-UA security [11]. The report concludes that message injection and tampering attacks constitute the most serious threats to I4.0 applications. Given this security assessment, we propose intruder models that capture these attack capabilities. More specifically, for general security verification we follow the Dolev–Yao intruder model (DY) [7], in which the intruder controls the entire network, and adapt it to our particular setting. Our intruder models represent intruders that can inject, manipulate and block messages. In addition to unbounded intruder models, we also consider bounded versions of intruders that can only perform a bounded number of intrusions on the system.

We then use these intruder models to study the Security Problem for Functionally Correct Systems (SP-FCS), which is to determine whether a functionally correct system can reach a bad configuration in the presence of an intruder. Some of the results obtained are summarized in Table 1. Our computational complexity results refer to standard complexity classes NP (non-deterministic polynomial time) and PSPACE (polynomial space) [21]. Table 1 differs from [3, Table 1] as it is expanded to include the new results for 1-bounded PAS.

In addition to the complexity results shown in Table 1, in Sections "Functional Correctness" and "Intruder Model" we also provide the complexity results for the parameterized version of the safety problem for LAS, where the bound on the number of applications of each of the automata instructions within one system cycle is taken as an additional parameter.

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 [Theorems 5.2, 5.3]	PSPACE-Complete [Theorems 6.5, 6.6]	PSPACE-Complete [Theorems 6.5, 6.6]
PAS	PSPACE-Complete [Theorems 5.2, 5.3]	PSPACE-Complete [Theorems 6.5, 6.6]	PSPACE-Complete [Theorems 6.5, 6.6]
1-bounded PAS	coNP-Complete [Corollary 5.6, Theorem 5.7]	PSPACE-Complete [Theorems 6.5, 6.6]	PSPACE-Complete [Theorems 6.5, 6.6]

Even though the AS model is relatively simple, the complexity of both safety and security problems for AS is PSPACE-complete. In our investigations, we identified a class of AS, LAS, for which the complexity of FCP is co-NP-complete. However, the complexity of SP-FCS does not improve, even in the case when the intruder may only use one action.

This paper extends the conference paper [3] in several ways, while following its general presentation structure. We present in this paper new complexity results for 1-bounded PAS and we add full, detailed proofs of all the complexity results obtained. The paper also provides a complete and detailed treatment of the examples of automata systems and the different attacks. Finally, the description of the automated verification has been considerably expanded w.r.t. [3], including the description of the methodology and experiments.

Sections "Related Work" and "Motivating Example" motivate this work with related work and an example taken from an I4.0 application. In Section "Formal Model" we introduce AS as an MSR model and specify various classes of AS. In Section "Functional Correctness" we define FCP and prove complexity results for various AS classes. In Section "Intruder Model" we introduce MSR intruder models and present complexity results for SP-FCS for different assumptions about intruders and types of systems. Section "Detailed Proofs of Complexity Results" contains detailed proofs of the complexity results. In Section "Automated Verification" we present results of SP-FCS experiments obtained using Maude. We conclude in Section "Conclusions and Future Work" by pointing to future work.

Related Work

Since I4.0 systems include a variety of interconnected components, including the Internet of Things and cyber-physical devices, the exhaustive analysis of I4.0 systems should consider the specific characteristics of these elements. In [17], methods for the verification of cyber-physical systems that take into account their actual physical behavior have recently been proposed. The approach of this paper, on the other hand, does not address such details, but only considers an abstract level. This has a major impact on the type of verification that is performed. However, the formal model of this work allows for extensions that would address relevant aspects, such as time [15, 22].

The work in [17] approaches verification of I4.0 through statistical model checking. The approach in [17] combines the formal executable specification of I4.0 applications with a bounded intruder model. This is achieved using the rewriting-logic-based tool Maude [6] and its rewrite modules. It is shown that such a bounded intruder is already capable of doing damage to the system by injecting messages that are then received at the wrong time, violating safety invariants and reaching a bad state of the system. For an equationally defined bad state of the system, the possible attack scenarios can be enumerated using Maude's search capability.

In this paper, we use the symbolic approach of [19] developed for formal analysis of security of I4.0 applications. Their abstract application and intruder models enable symbolic execution in Maude and demonstrate how the search can find all attack points, which can then provide information for system designers to protect the message flow in the application with signatures.

The symbolic approach can be combined with abstraction techniques, such as [20]. In particular, the abstraction techniques used in the framework of [20] support the engineering design workflow using theory transformations.

For a given map illustrating the deployment from application components to devices, a theory transformation is defined. The theory models the execution of the application on the given set of networked devices. For an enumeration of message flows that represent attacks, another theory transformation is defined that provides a security wrapper with signature verification policies for each device [20]. This paper, together with the conference paper [3] it extends, provides a mathematical basis for the specification framework in [19] that is executable in Maude.

Motivating Example

In this paper, we use as a running example a simple I4.0 unit called Pick and Place (PnP),[1] of which an earlier version appeared in previous work [3]. Figure 1 shows the architecture of the PnP application. This is a common pattern in production lines, where a mechanical arm picks up an object from one location and places it at another. In this example, a conveyor belt brings containers to a barrier point, where there is a source of caps for the containers. The arm moves along a track to the right and positions itself over the caps. It turns on its vacuum mechanism to lift a cap and moves left along the track until it is positioned over a container. It turns off the vacuum, releases the cap and is ready for the next cycle.

The PnP application model consists of three function blocks:

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

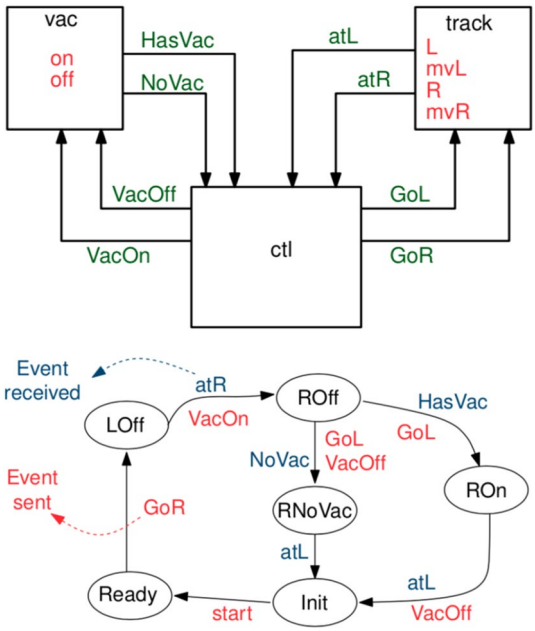


Fig. 1 (Top) PnP Function Blocks, ctl, vac, and track, where the internal states of vac and track are shown in their corresponding boxes and their transitions are elided. (Bottom) The complete specification of ctl as a finite-state machine. [3, Figure 1]

track—the FB that controls the movement of the arm;
vac—the FB that controls the on/off state of the vacuum mechanism and also contains a sensor for detecting whether the piece has been picked up; and
ctl—the FB that coordinates the track and vac FBs.

These FBs communicate according to the connections shown in Fig. 1 on the left. The behavior of an FB is specified by an interactive automaton similar to a Mealy automaton. Transitions change the states of the FBs and represent the communication between the FB. More precisely, transitions are guarded by predicates for incoming signals (called events). When a transition is executed, outgoing events are generated and transformed into incoming events according to the network connections.

The automaton representing the ctl FB is shown on the right in Fig. 1. Starting from its initial state, Init, the ctl automaton sends the message start to itself to start a new production cycle and changes the state to Ready. Then it sends a GoR event and changes state to LOff, which is the state of the PnP system when the arm is positioned at the left end and the vacuum mechanism is off. The GoR signal instructs the track automaton to move the arm to the right. Once the arm has moved all the way to the right, the track automaton confirms this with an atR event. When an atR signal arrives, the ctl automaton moves to ROff state, which denotes that the arm is positioned at the right end and the vacuum is off. Then the ctl automaton sends the VacOn signal to the vac automaton to turn the vacuum on. The controller proceeds similarly, moving the cap from the right side to the left side to place it in the correct position. It then places the cap on the container by deactivating the vacuum. If the vacuum mechanism fails to pick a cap, it sends the NoVac event to the ctl automaton. In this case, the ctl automaton moves to the RNoVac state and sends a signal to the vac automaton to de-activate the vacuum pump and a signal to the track automaton to move the arm to the left side of the PnP.

This application is a typical manufacturing application, where a task is periodically repeated. This system property is reflected in the fact that all cycles in the FB specification contain the initial state of the automata.

To ensure the safety of the system, the analysis of the logical behavior of the system is usually performed using methods, such as Systems Theoretic Process Analysis (STPA) [18]. The analysis should determine which system configurations are bad in the sense that they pose a safety risk and should, therefore, be avoided.

In the case of the PnP example, a safety risk is that a cap falls off while it is being moved. One can imagine that the PnP unit does not put caps on containers, but instead handles heavy objects, e.g., heavy bricks. Premature release of the object could injure someone in the vicinity or damage the factory itself, e.g., damage the conveyor belt. This safety risk is related to the *critical configuration*, which is specified by the track automaton being in the mvL state, the vac automaton being in the off state, and the ctl automaton being in the ROn or Init state. That is, the ctl automaton has received the signal that the vac automaton has picked up the cap, but as it moves to the left, the state of the vac automaton is off, indicating that the cap has been released. One way such a bad configuration could be reached is to have the ctl automaton send a VacOff event before the arm is positioned all the way to the left (perhaps to optimize something). If it were possible for the PnP application to apply the transitions of its FBs and reach the critical configuration specified above, the PnP application would represent an I4.0 application that is not *functionally correct*.

Furthermore, for a functionally correct I4.0 application, we investigate whether an intruder capable of injecting an event into any one of the connections at any point in time can bring the application into a critical configuration. In the PnP application example, the answer is yes. As described in [19], there are in fact four ways the intruder can do this. For example, while the cap is being moved, the attacker can send the message VacOff to vac, causing it to release the cap. Alternatively, the attacker can send the message atL to the ctl automaton even though the cap is still being moved, causing the ctl automaton to deactivate the vacuum pump prematurely and release the cap.

Formal Model

We briefly review Multiset Rewriting (MSR) models of [13], which is the language we use to specify systems that model I4.0 applications and intruders.

Multiset Rewriting Systems

Assume a finite typed first-order alphabet, Σ, with variables, constants, function and predicate symbols. Terms and facts are constructed by applying symbols of correct type (see [9]). For instance, if P is a predicate of type $\tau_1 \times \tau_2 \times \cdots \times \tau_n \rightarrow o$, 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. Intuitively, a configuration describes a state of a system. Configurations are modified by multiset rewrite rules. Rewrite rules model processes, which can be interpreted as actions of the system. *Actions* are multiset rewrite rules:

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

While the facts $W_1, \ldots, W_k$ are preserved by the above rule, the facts $F_1, \ldots, F_n$ are replaced by the facts $Q_1, \ldots, Q_m$. All free variables that occur in the postcondition must also occur in the precondition of the rule. 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 suitable substitution θ, such that $\mathcal{S}_0 = \mathcal{W}\theta$. The configuration that results from applying this rule to $\mathcal{S}$ is $(\mathcal{S} \setminus \mathcal{S}_0) \cup (\mathcal{W}'\theta)$. (The application of substitution $(\mathcal{S}\theta)$ is defined by the mapping of term variables to terms.)

Definition 4.1 (*Trace*) A *trace* of MSR rules $\mathcal{R}$ from a given configuration $\mathcal{S}_0$ 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 $\mathcal{S}_i$.

It is in nature of multiset rewriting that there are different aspects of non-determinism in the model. For example, different rules or different instantiations of the same rule may be applicable to the same configuration, resulting in different configurations.

The main MSR problems, reachability problems, reduce to the existence of traces over given rules from a given initial configuration to a given configuration. Since reachability problems are undecidable in general [16], some restrictions are imposed on the form and application of rules to achieve decidability. Of particular interest are MSR systems with *balanced* rules, i.e., rules in which the total number of facts occurring in the precondition and in the postcondition is the same. Systems containing only balanced rules constitute an important class of *balanced systems*, for which several reachability problems have been shown to be decidable [13, 15, 16].

In this paper, we use only balanced MSR systems. Our MSR systems, representing I4.0 applications and intruders, are balanced and denote a fixed setting of function blocks that communicate with a fixed set of signals over a network with a fixed capacity.

Industry 4.0 Specifications as MSR Models

We now show how the systems described in Section "Motivating Example" that model I4.0 applications are specified as formal MSR models. The underlying signature contains a finite number of each of the following symbols:

- constants denoting automata states;
- constants denoting signals;
- constant $*$ used for denoting an emtpy channel;
- predicates denoting the states of each automaton A, Q_A, and
- predicates denoting channels, $R_{A,B}$.

We define *automata systems* (AS) representing a network of FBs. Automata in an AS are conceived as event-driven finite automata that communicate by exchanging a fixed set of (atomic) signals over a fixed number of distinct channels. Automata in an AS are defined by a finite set of balanced rules that specify how the received event signals prompt the automata to act.

Some of the automata, say A and B, can interact directly through a channel to which A can write and from which B read/consume. We denote such a channel by the predicate $R_{A,B}$. Given a channel $R_{A,B}$, the fact $R_{A,B}(\mathbb{m})$ denotes that using $R_{A,B}$, 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 $R_{A,B}$ is empty. One interpretation is that $R_{A,B}$ is a channel with a single-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 $\mathcal{R}$ is 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 that does not contain the $*$ symbol, and X_A is a finite set of *instructions* of the form:

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

Fig. 2 Instructions of PnP AS [3, Figure 2]

$$
\begin{aligned}
&Q_A(\mathtt{q}),\ R_{B_1,A}(\mathtt{m}_1),\ldots,R_{B_k,A}(\mathtt{m}_\mathtt{k}),\\
&R_{A,C_1}(*),\ldots,R_{A,C_\ell}(*) \ \longrightarrow\\
&\qquad Q_A(\mathtt{q}'),\ R_{B_1,A}(*),\ldots,R_{B_k,A}(*),\\
&R_{A,C_1}(\mathtt{m}'_1),\ldots,\ R_{A,C_\ell}(\mathtt{m}'_\ell),
\end{aligned}
\tag{1}
$$

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

We refer to all the instructions of automata in $\mathcal{N}=(\mathcal{A},\mathcal{R})$ as *system rules* $X_\mathcal{N}$, that is $X_\mathcal{N}=\bigcup_{A\in\mathcal{A}} X_A$.

According to the rule, Eq. (1), when each of the channels R_{A,C_j}, $1\le j\le \ell$, is free, the automaton A, which is in state q and receives the signals $\mathtt{m}_1,\ldots,m_k$ through each of the channels $R_{B_1,A},\ldots,R_{B_k,A}$, respectively, moves to its state q', provides the signals $\mathtt{m}'_1,\ldots,m'_\ell$ through each of the channels $R_{A,C_1},\ldots,R_{A,C_\ell}$, respectively. In doing so, it discharges the signals $\mathtt{m}_1,\ldots,m_k$, thereby releasing all the channels $R_{B_1,A},\ldots,R_{B_k,A}$. In the special case when $k=0$, the rule Eq. (1) denotes the action of the automaton, which is internally triggered only by the state of the automaton.

As an example, consider the PnP automata system shown in Fig. 1, $\mathcal{N}_{\mathsf{PnP}}=(\mathcal{A}_{\mathsf{PnP}},\mathcal{R}_{\mathsf{PnP}})$, which consists of three function blocks, $\mathcal{A}_{\mathsf{PnP}}=\{\mathsf{ctl},\mathsf{track},\mathsf{vac}\}$ and the set of communication channels $\mathcal{R}_{\mathsf{PnP}}=\{R_{\mathsf{ctl},\mathsf{vac}}(*),R_{\mathsf{ctl},\mathsf{track}}(*),R_{\mathsf{track},\mathsf{ctl}}(*),R_{\mathsf{vac},\mathsf{ctl}}(*),R_{\mathsf{ctl},\mathsf{ctl}}(*)\}$. The automata are defined as follows:

$$
\begin{aligned}
\mathsf{ctl} &= \big(S_{\mathsf{ctl}},\mathsf{Init},M_{\mathsf{ctl}},X_{\mathsf{ctl}}\big),\\
&\quad S_{\mathsf{ctl}}=\{\mathsf{Init},\mathsf{Ready},\mathsf{LOff},\mathsf{ROff},\mathsf{ROn},\mathsf{RNoVac}\},\\
&\quad M_{\mathsf{ctl}}=\{\mathsf{start},\mathsf{GoL},\mathsf{GoR},\mathsf{atL},\mathsf{atR},\mathsf{VacOn},\mathsf{VacOff},\\
&\qquad\quad \mathsf{NoVac},\mathsf{HasVac}\},\\
\mathsf{track} &= \big(S_{\mathsf{track}},\mathsf{L},M_{\mathsf{track}},X_{\mathsf{track}}\big),\\
&\quad S_{\mathsf{track}}=\{\mathsf{L},\mathsf{R},\mathsf{mvR},\mathsf{mvL}\},\\
&\quad M_{\mathsf{track}}=\{\mathsf{GoL},\mathsf{GoR},\mathsf{atL},\mathsf{atR}\},\\
\mathsf{vac} &= (S_{\mathsf{vac}},\mathsf{VacOff},M_{\mathsf{vac}},X_{\mathsf{vac}}),\\
&\quad S_{\mathsf{vac}}=\{\mathsf{off},\mathsf{on}\},\\
&\quad M_{\mathsf{vac}}=\{\mathsf{VacOn},\mathsf{VacOff},\mathsf{NoVac},\mathsf{HasVac}\},
\end{aligned}
\tag{2}
$$

where the corresponding system rules of $\mathcal{N}_{\mathsf{PnP}}$ are specified in Fig. 2. In particular, the instruction $\mathtt{c}_3$ denotes the following action of the ctl automaton: being in the state LOff and getting the signal atR denoting that the arm is in the rightmost position, the ctl automaton sends the signal VacOn to engage the vacuuming action with the vacuum device, i.e., the automaton vac.

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 a system configuration contains exactly one of each of the channel predicates R_{A_i,A_j}, we model systems with at most one channel from one automaton in the system to another, where each channel has a single buffer capacity.

The assumption of a single buffer capacity typically occurs in many I4.0 applications, especially in (parts of) applications that require high performance or are safety critical [1]. This is implemented using message delivery schedules, such as those in Time Sensitive Networks, so that only one message is received and processed at a time. However, it is possible to extend our model to represent multiple channels and larger network capacities, e.g., using multiple $R_{A,B}$ facts in the configuration, each representing a single channel buffer, or using special facts denoting network bandwidth. However, the implications of such extensions on the complexity are reserved for future work.

The initial configuration of an AS denotes the system configuration at the beginning of the production process, i.e., with all automata in the initial state and with all channels free.

Definition 4.4 (*Initial Configuration*) *Initial configuration* of an AS $\mathcal{N}=(\mathcal{A},\mathcal{R})$ is the system configuration of $\mathcal{N}$ which consists of exactly 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 exactly one fact $\mathcal{R}_{A_i,A_j}(*)$ for each channel $\mathcal{R}_{A_i,A_j} \in \mathcal{R}$.

For example, the initial configuration of PnP AS is the following:

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

Some of the system configurations represent bad overall states of systems which should be avoided. Such system configurations represent situations that are undesired w.r.t. functionality of the I4.0 application being modelled by the AS and are called *critical configurations*.

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.

For example, for the PnP system shown in Fig. 1, it is critical for the vacuum to switch off, while the arm is moving to the left and carrying a cap. Consequently, any PnP system configuration containing either the facts

$$\{Q_{\mathsf{vac}}(\mathsf{off}),\ Q_{\mathsf{track}}(\mathsf{mvL}),\ Q_{\mathsf{ctl}}(\mathsf{ROn})\} \tag{3}$$

or the facts

$$\{Q_{\mathsf{vac}}(\mathsf{off}),\ Q_{\mathsf{track}}(\mathsf{mvL}),\ Q_{\mathsf{ctl}}(\mathsf{Init})\} \tag{4}$$

would be critical.

We assume that each AS has an associated specification of critical configurations. Recall that for I4.0 applications, critical configurations are usually determined using methods, such as Systems Theoretic Process Analysis [18].

Given that I4.0 applications are written as Mealy machines, one might question the motivation for using MSR models. One reason for using MSR models is that it is straightforward to add intruder models and define the corresponding verification problems. This is described in Section 6. In contrast, Mealy machines are not suitable for specifying intruders that can send messages at any time in any of the channels. Another reason for using MSR theories is that MSR rules are more general and can be used to express more features, such as nonces used in protocol security research, which are not available in Mealy machines. Although nonces and cryptographic protocols are not generally used in this work, our models can be easily extended to formalize such features, e.g., signed messages, see [13].

Periodic Automata Systems

We introduce specific subclasses of AS by incorporating further requirements of I4.0. One of these assumptions is that a typical I4.0 application is periodic, that is, a collection of its tasks is repeated over and over again. For example, the PnP application described in Section 3, repeats the task of placing a cap over a container.

In I4.0 terminology, FBs operate in *micro-cycles*, in which each FB repeats one of its cycles, while the entire application operates in *hyper-cycles*, which begin and end in a system configuration in which all FBs are in their initial state.

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 system 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 such periodic behavior, we introduce a class of automata systems called Periodic Automata Systems (PAS) in which constraints are imposed on the system behaviour.

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

For example, the PnP in Fig. 1 is a PAS. One of its hyper-cycles models the placement of a cap over a container. Each of the automata in the PnP application takes part in this process. In particular, the ctl automaton can run through its initial state Init in two different cycles, the outer one modelling the successful placement of a cap, and the inner one in which the vacuum pump fails to pick up a cap, so the system must reset for the next round.

Rules $\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3, \mathsf{c}_4, \mathsf{c}_5$ formalize the outer cycle of the ctl automaton, while rules $\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3, \mathsf{c}_6, \mathsf{c}_7$ formalize the other cycle of the ctl automaton, as illustrated in Fig. 1.

Proposition 4.8 *Given a* PAS $\mathcal{N}$, *a system configuration of* $\mathcal{N}$ *is reachable from an initial configuration of* $\mathcal{N}$ *if and only if it is reachable within one hyper-cycle.*

Proof Let P be a finite trace of $\mathcal{N}$ from the initial configuration of $\mathcal{N}$, $\mathcal{S}_0$, to the configuration $\mathcal{S}$ of $\mathcal{N}$. By the definition of PAS, any finite trace of rules of a PAS from its initial configuration can be extended to an infinite trace. Moreover, all its infinite traces are concatenations of hyper-cycles. In other words, every finite trace of $\mathcal{N}$ from the initial configuration

of $\mathcal{N}$ is a prefix of a concatenation of hyper-cycles of $\mathcal{N}$. Therefore, the configuration $\mathcal{S}$ belongs to a hyper-clycle H of $\mathcal{N}$. Then a trace P', defined as the prefix of H ending in $\mathcal{S}$ is a trace from $\mathcal{S}_0$ to $\mathcal{S}$ showing that the configuration $\mathcal{S}$ is reachable from the initial configuration of $\mathcal{N}$ within one hyper-cycle.

Since each hyper-cycle starts with the initial configuration, the converse implication also holds. □

One must keep in mind that within any hyper-cycle of a PAS the number of applications of instructions of any automaton could in principle be exponential. On the other hand, in the PnP example shown in Fig. 1 each of the automata instructions is applied at most once in a PnP hyper-cycle. Assuming a bound on the number of applications of automata instructions within a hyper-cycle leads to the another class of AS.

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.

Note that by Definition 4.9 the PnP AS shown in Fig. 1 is a LAS, more precisely a 1-bounded PAS.

Functional Correctness

Functional correctness is a safety property for AS. It is an unreachability problem over system rules, where the critical configurations are specified over states of FBs, denoting bad configurations of the system.

Definition 5.1 (*Functional Correctness*) An automata system $\mathcal{N}$ is *functionally correct* if there is no trace of $\mathcal{N}$ leading from the initial configuration of $\mathcal{N}$ to a critical configuration of $\mathcal{N}$.

The Functional Correctness Problem (FCP) is the problem of determining whether a given AS is functionally correct.

Functionally correct systems guarantee the correct execution of the working process, within the closed system, without interference from outside. Nevertheless, this is not a guarantee of security, since actions of intruders can lead to undesired system configurations.

For example, the PnP AS shown in Fig. 1 is functionally correct. Namely, no critical configuration, as specified in Eqs. 3)–(4), is reachable from the initial configuration using the system rules. Indeed, the only hypes-cycles of PnP are the following sequences of system rules:

$c_1, c_2, t_1, t_2, c_3, v_1, c_4, t_3, t_4, c_5, v_3$;
$c_1, c_2, t_1, t_2, c_3, v_2, c_6, t_3, t_4, v_3$;
$c_1, c_2, t_1, t_2, c_3, v_2, c_6, t_3, v_3, t_4$;
$c_1, c_2, t_1, t_2, c_3, v_2, c_6, v_3, t_3, t_4$.

By inspection, none of the system configurations reached on the above traces, i.e., within one hyper-cycle, is critical. Hence, according to Proposition 4.8, no critical configuration of PnP AS is reachable from the initial configuration on any trace of system rules.

However, as we show in Section 6.1, the PNP AS is not secure in the presence of an intruder with access to its communication channels.

The complexity of the functional correctness problem for AS may involve exponentially long traces, since even minimal hyper-cycles may be exponentially long.

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 over each channel. This number is exponential in the number of automata and channels within the system.

The following theorem provides a PSPACE upper bound for the FCP for the case of general AS.

Theorem 5.2 (An upper bound for FCP for AS) *For general* AS, *functional correctness problem*, FCP, *is in PSPACE.*

Proof Given an AS $\mathcal{N}$, we take into account that the number of channels and their capacity are fixed in advance. Therefore, we fix the number of facts denoting the current states of the automata and the content of the interface channels. Moreover, the total number of symbols contained in any system configuration of $\mathcal{N}$ is polynomial in $\mathcal{N}$.

Therefore, any trace that is representing an appropriate sequence of actions from the initial configuration to a critical configuration, can be guessed and checked in NPSPACE [16].

Accordingly, functional correctness problem, that is, verifying that no critical/bad configuration is reachable, belongs to co-NPSPACE. Since NPSPACE, co-NPSPACE, and PSPACE are the same complexity classes, this provides the PSPACE upper bound for FCP for general AS. □

In the next theorem we provide a lower bound for the FCP for AS, even for periodic automata systems.

Theorem 5.3 (A lower bound for FCP for PAS) *Functional correctness problem is PSPACE-hard for* AS, *even for* PAS.

Proof (Proof Sketch) To obtain the lower bound for FCP, we simulate deterministic Turing machines running in PSPACE using PAS.

The main challenge we address here, among others, is that within Turing computations we are dealing with a stable device, a tape, for permanent storage of the required information. In our automata approach, the situation is the opposite, namely, every time we read the signal m stored in the channel $R_{A,B}(m)$, we have to nullify, i.e., empty, the channel $R_{A,B}$.

The encoding and corresponding properties are quite intricate. For the sake of readability, we omit all details here. The detailed proof can be found in Section 7.1. □

When considering a k-bounded PAS, it makes sense to investigate a parameterized version of the FCP problem in which the bound k is considered as an additional part of the input to the decision problem.

Definition 5.4 (*k-bounded Functional Correctness Problem for* LAS) Given a bound k, the *k-bounded Functional Correctness Problem for* LAS (k-FCP) is the problem of determining whether a given k-bounded PAS is functionally correct.

Considering k as an additional part of the input of the FCP problem for LAS affects the complexity. Namely, for the complexity of the k-FCP we obtain the co-NP upper bound.

Theorem 5.5 (An upper bound for k-FCP) *The k-bounded functional correctness problem for* LAS *is in coNP.*

Proof Let $\mathcal{N} = (\mathcal{A}, \mathcal{R})$ be a k-bounded PAS. The number of system actions used in a single hyper-cycle is polynomial in the size of $\mathcal{N}, k$. Therefore, an appropriate sequence of actions leading from the initial configuration to a critical configuration can be guessed in NP.

That a system is functionally correct means that such a "bad" sequence of actions within the system is impossible. Therefore, FCP belongs to coNP. □

From the above result of the parameterized version of the FCP problem we immediately obtain an upper bound for the original FCP for the class of 1-bounded PAS.

Corollary 5.6 (An upper bound for FCP for 1-bounded PAS) *The functional correctness problem for* 1*-bounded* PAS *is in coNP.*

For the class of LAS, the FCP turns out to be coNP-hard, even in the restricted case of 1-bounded PAS, where each of the instructions in the system may be applied at most once in a hyper-cycle.

Theorem 5.7 (A lower bound for FCP for 1-bounded PAS) *The functional correctness problem for* 1*-bounded* PAS *is coNP-hard.*

This result is obtained by encoding the 3-SAT problem using automata systems. The AS used in the encoding is a 1-bounded PAS. A detailed proof is given in Section 7.2.

Following Definition 4.9, the parameter k in the k-FCP problem is an upper bound on the number of applications of instructions of any automata in the PAS within a hyper-cycle. Since the automata instructions of the PAS from the encoding used in the proof of Theorem 5.7 are applied at most once in a hyper-cycle, the encoding also provides the following result.

Corollary 5.8 (A lower bound for k-FCP) *The k-bounded functional correctness problem for* LAS *is coNP-hard.*

Bringing together the Corollary 5.8 and Theorem 5.5, we can conclude that the k-FCP is coNP-complete.

Theorem 5.9 *The k-bounded functional correctness problem for* LAS *is coNP-complete.*

Intruder Model

In this section, we present an intruder model relevant to I4.0. It is based on the Dolev–Yao intruder model [7], but has been adapted to I4.0 applications considering the findings of the BSI security assessment [11] of OP-CUA. The assessment concludes that the greatest threats to I4.0 applications come from the injection and tampering of messages. Our intruder model incorporates these capabilities, and, in addition, supports the intruder's ability to block messages.

The DY intruder models, such as the ones in [8, 13, 22, 23] include various intruder capabilities, in particular intruder rules for message pairing, encryption, and decryption. Unlike such models and their general application, such intruder rules do not contribute to the power of intruder here. Note that messages communicated in channels of automata systems are not encrypted. In other words, the intruder does not need to eavesdrop to collect some knowledge about the system. Instead, the intruder already knows all possible messages that can be exchanged in an AS. This is formalised by assuming that the intruder is familiar with all the signal constants that can be exchanged between automata in the system.

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

Definition 6.1 (*Intruder*) An *intruder* $\mathcal{I}$ is represented as a one state automaton which is defined by a finite set of rules $\mathcal{R}_{\mathcal{I}}$ of the form:

$$R_{A,C}(*) \longrightarrow R_{A,C}(\mathtt{m}) \tag{5}$$

$$R_{A,C}(\mathtt{m}) \longrightarrow R_{A,C}(*) \tag{6}$$

$$R_{A,C}(\mathtt{m}) \longrightarrow R_{A,C}(\mathtt{m}'), \tag{7}$$

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

Remark 6.2 Since automata representing intruders have only one state, for simplicity we abbreviate the form of Eq.(1) by omitting the facts that denote the automata states in rules given in Eqs.(5), (6) or (7).

Using the rule Eq. (5), an intruder injects a signal $\mathtt{m}$ into an empty channel. Using the rule Eq. (6), an intruder removes a signal $\mathtt{m}$ from a channel while using the rule Eq. (7), an intruder modifies a signal $\mathtt{m}$ in a channel into a signal $\mathtt{m}'$.

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

Definition 6.3 (*Bounded Intruder*) A *bounded intruder* $\mathcal{I}$ is an intruder that, when interfering with a AS $\mathcal{N} = (\mathcal{A}, \mathcal{R})$, is allowed to use actions of type Eqs. (5), (6), or (7) on any channel $R_{A,C} \in \mathcal{R}$ using some signal(s) $\mathtt{m}, \mathtt{m}' \in M_A$, $A \in \mathcal{A}$ only a bounded number of times.

There are several motivations to consider bounded intruders. One of the motivations comes from the I4.0 applications themselves. As discussed in [17], I4.0 applications are Cyber-Physical systems, where every action takes time, including the actions of the intruder. This means that the intruder cannot perform an unbounded number of actions within a given time period. This is similar to notions of progressing systems [12]. A second analogy and motivation is that bounded intruder models correspond to bounded verification problems, such as the processes considered in bounded model checking [4].

We describe below in a detailed way example attacks that are performed by the intruder defined above, which were first presented in an abbreviated form in [3].

Example Attack by Message Insertion on PnP AS

In this section, we describe an attack on the Pick and Place automata system illustrated in Fig. 1. Recall from Section 4.2 that it is critical for the PnP system that the vacuum is switched off, while the arm is moving to the left. This bad situation is denoted by a configuration that includes the facts $Q_{\mathsf{vac}}(\mathtt{off})$, $Q_{\mathtt{track}}(\mathsf{mvL})$, $Q_{\mathsf{ctl}}(\mathsf{ROn})$. Recall also that the PnP AS is functionally correct.

We show that any intruder with access to channel $R_{\mathsf{ctl,vac}}$ is able to perform the attack on this AS. Starting from the initial configuration:

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

a consecutive application of the following system rules $\mathtt{c}_1$, $\mathtt{c}_2$, $\mathtt{t}_1$, $\mathtt{t}_2$, $\mathtt{c}_3$, $\mathtt{v}_1$, $\mathtt{c}_4$, $\mathtt{t}_3$ is possible:

$$Q_{\mathsf{ctl}}(\mathsf{Init}), Q_{\mathsf{track}}(\mathsf{L}), Q_{\mathsf{vac}}(\mathtt{off}), \\ R_{\mathsf{ctl,vac}}(*), R_{\mathsf{ctl,track}}(*), R_{\mathsf{track,ctl}}(*), R_{\mathsf{vac,ctl}}(*), \\ R_{\mathsf{ctl,ctl}}(*) \longrightarrow_{c_1}$$

$$Q_{\mathsf{ctl}}(\mathsf{Ready}), Q_{\mathsf{track}}(\mathsf{L}), Q_{\mathsf{vac}}(\mathtt{off}), \\ R_{\mathsf{ctl,vac}}(*), R_{\mathsf{ctl,track}}(*), R_{\mathsf{track,ctl}}(*), R_{\mathsf{vac,ctl}}(*), \\ R_{\mathsf{ctl,ctl}}(\mathsf{start}) \longrightarrow_{c_2}$$

$$Q_{\mathsf{ctl}}(\mathsf{LOff}), Q_{\mathsf{track}}(\mathsf{L}), Q_{\mathsf{vac}}(\mathtt{off}), \\ R_{\mathsf{ctl,vac}}(*), R_{\mathsf{ctl,track}}(\mathsf{GoR}), R_{\mathsf{track,ctl}}(*), R_{\mathsf{vac,ctl}}(*), \\ R_{\mathsf{ctl,ctl}}(*) \longrightarrow_{t_1}$$

$$Q_{\mathsf{ctl}}(\mathsf{LOff}), Q_{\mathsf{track}}(\mathsf{mvR}), Q_{\mathsf{vac}}(\mathtt{off}), \\ R_{\mathsf{ctl,vac}}(*), R_{\mathsf{ctl,track}}(*), R_{\mathsf{track,ctl}}(*), R_{\mathsf{vac,ctl}}(*), \\ R_{\mathsf{ctl,ctl}}(*) \longrightarrow_{t_2}$$

$$Q_{\mathsf{ctl}}(\mathsf{LOff}), Q_{\mathsf{track}}(\mathsf{R}), Q_{\mathsf{vac}}(\mathtt{off}), \\ R_{\mathsf{ctl,vac}}(*), R_{\mathsf{ctl,track}}(*), R_{\mathsf{track,ctl}}(\mathsf{atR}), R_{\mathsf{vac,ctl}}(*), \\ R_{\mathsf{ctl,ctl}}(*) \longrightarrow_{c_3}$$

$$Q_{\mathsf{ctl}}(\mathsf{ROff}), Q_{\mathsf{track}}(\mathsf{R}), Q_{\mathsf{vac}}(\mathtt{off}), \\ R_{\mathsf{ctl,vac}}(\mathsf{VacOn}), R_{\mathsf{ctl,track}}(*), R_{\mathsf{track,ctl}}(*), R_{\mathsf{vac,ctl}}(*), \\ R_{\mathsf{ctl,ctl}}(*) \longrightarrow_{v_1}$$

$$Q_{\mathsf{ctl}}(\mathsf{ROff}), Q_{\mathsf{track}}(\mathsf{R}), Q_{\mathsf{vac}}(\mathsf{on}), \\ R_{\mathsf{ctl,vac}}(*), R_{\mathsf{ctl,track}}(*), R_{\mathsf{track,ctl}}(*), R_{\mathsf{vac,ctl}}(\mathsf{HasVac}), \\ R_{\mathsf{ctl,ctl}}(*) \longrightarrow_{c_4}$$

$$Q_{\mathsf{ctl}}(\mathsf{ROn}), Q_{\mathsf{track}}(\mathsf{R}), Q_{\mathsf{vac}}(\mathsf{on}), \\ R_{\mathsf{ctl,vac}}(*), R_{\mathsf{ctl,track}}(\mathsf{GoL}), R_{\mathsf{track,ctl}}(*), R_{\mathsf{vac,ctl}}(*), \\ R_{\mathsf{ctl,ctl}}(*) \longrightarrow_{t_3}$$

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

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

Fig. 3 Instructions of the Example LAS [3, Figure 3]

At that point, an intruder inserts a signal into the channel from the ctl automaton to the vac automaton using the intruder rule of type Eq (5): $R_{\mathsf{ctl},\mathsf{vac}}(*) \longrightarrow R_{\mathsf{ctl},\mathsf{vac}}(\mathsf{VacOff})$, obtaining the configuration:

$$
\begin{array}{l}
Q_{\mathsf{ctl}}(\mathsf{ROn}), Q_{\mathsf{track}}(\mathsf{mvL}), Q_{\mathsf{vac}}(\mathsf{on}), \\
\quad R_{\mathsf{ctl},\mathsf{vac}}(\mathsf{VacOff}), R_{\mathsf{ctl},\mathsf{track}}(*), R_{\mathsf{track},\mathsf{ctl}}(*), R_{\mathsf{vac},\mathsf{ctl}}(*), R_{\mathsf{ctl},\mathsf{ctl}}(*)\,.
\end{array}
$$

To the obtained configuration, the system rule $\mathtt{v}_2$ is applicable. This leads to the critical configuration:

$$
\begin{array}{l}
Q_{\mathsf{ctl}}(\mathsf{ROn}), Q_{\mathsf{track}}(\mathsf{mvL}), Q_{\mathsf{vac}}(\mathtt{off}), \\
\quad R_{\mathsf{ctl},\mathsf{vac}}(*), R_{\mathsf{ctl},\mathsf{track}}(*), R_{\mathsf{track},\mathsf{ctl}}(*), R_{\mathsf{vac},\mathsf{ctl}}(\mathsf{NoVac}), R_{\mathsf{ctl},\mathsf{ctl}}(*).
\end{array}
$$

The above attack shows that any intruder with the capability of injecting a signal into empty channels of PnP PAS can lead the system to a bad state by a single message injection. This means that a functionally correct system, even a PAS, might not be secure in the presence of an intruder.

Example Attack on a LAS: Breaking a Hyper-cycle

We now present an attack on an AS that shows that an intruder can even change the nature of the system so that a PAS no longer behaves periodically.

Consider the following example of an AS. Let $\mathcal{N} = (\mathcal{A}, \mathcal{R})$, where

$$
\begin{array}{l}
\mathcal{A} = \{A_1, A_2, A_3\}, \\
\mathcal{R} = \{R_{A_1,A_2}, R_{A_2,A_1}, R_{A_2,A_2}, R_{A_1,A_3}, R_{A_3,A_1}\}, \\
A_1 = (S_1, q_0^1, M, X_1),\ A_2 = (S_2, q_0^2, M, X_2), \\
A_3 = (S_3, q_0^3, M, X_3), \\
S_1 = \{q_0^1, q_1^1, q_2^1, q_3^1\}, S_2 = \{q_0^2, q_1^2, q_2^2\}, S_3 = \{q_0^3, q_1^3\}.
\end{array}
$$

Let $M = \{*, \mathtt{a}, \mathtt{b}, \mathtt{c}\}$ be the set of signals of all automata in $\mathcal{N}$. Finally, let the set of instructions X_i of each automaton A_i, $1 \le i \le 3$, defined as in Fig. 3. Let critical configurations of $\mathcal{N}$ be those system configurations that contain the fact $Q_{A_1}(q_2^1)$.

The given AS is a functionally correct LAS. The only hyper-cycle of $\mathcal{N}$ is the following:

$$
\begin{array}{l}
\mathcal{S}_0 = Q_{A_1}(q_0^1),\ Q_{A_2}(q_0^2),\ Q_{A_3}(q_0^3), \\
\qquad R_{A_2,A_2}(*),\ R_{A_1,A_2}(*),\ R_{A_2,A_1}(*), \\
\qquad R_{A_1,A_3}(*),\ R_{A_3,A_1}(*) \longrightarrow_{s_1} \\
\quad Q_{A_1}(q_0^1),\ Q_{A_2}(q_1^2),\ Q_{A_3}(q_0^3), \\
\qquad R_{A_2,A_2}(\mathtt{b}),\ R_{A_1,A_2}(*),\ R_{A_2,A_1}(*), \\
\qquad R_{A_1,A_3}(*),\ R_{A_3,A_1}(*) \longrightarrow_{s_2} \\
\quad Q_{A_1}(q_0^1),\ Q_{A_2}(q_2^2),\ Q_{A_3}(q_0^3), \\
\qquad R_{A_2,A_2}(*),\ R_{A_1,A_2}(*),\ R_{A_2,A_1}(\mathtt{b}), \\
\qquad R_{A_1,A_3}(*),\ R_{A_3,A_1}(*) \longrightarrow_{r_1} \\
\quad Q_{A_1}(q_1^1),\ Q_{A_2}(q_2^2),\ Q_{A_3}(q_0^3), \\
\qquad R_{A_2,A_2}(*),\ R_{A_1,A_2}(*),\ R_{A_2,A_1}(*), \\
\qquad R_{A_1,A_3}(*),\ R_{A_3,A_1}(*) \longrightarrow_{r_2} \\
\quad Q_{A_1}(q_0^1),\ Q_{A_2}(q_2^2),\ Q_{A_3}(q_0^3), \\
\qquad R_{A_2,A_2}(*),\ R_{A_1,A_2}(\mathtt{a}),\ R_{A_2,A_1}(*), \\
\qquad R_{A_1,A_3}(*),\ R_{A_3,A_1}(*) \longrightarrow_{s_3} \\
\quad Q_{A_1}(q_0^1),\ Q_{A_2}(q_0^2),\ Q_{A_3}(q_0^3), \\
\qquad R_{A_2,A_2}(*),\ R_{A_1,A_2}(*),\ R_{A_2,A_1}(*), \\
\qquad R_{A_1,A_3}(*),\ R_{A_3,A_1}(*) = \mathcal{S}_0
\end{array}
$$

It consists of the consecutive application of rules s_1, s_2, r_1, r_2, s_3, given in Fig. 3. Notice that the hyper-cycle contains no rules of the automaton A_3.

Starting from the initial configuration $\mathcal{S}_0$, an infinite trace of $\mathcal{N}$ is obtained as the concatenation of this hyper-cycle. Hence, $\mathcal{N}$ is a periodic automata system. Moreover, each automaton rule is applied at most once in a hyper-cycle, hence $\mathcal{N}$ is an LAS.

LAS $\mathcal{N}$ is functionally correct since the critical configuration, that is, a configuration containing a fact $Q_{A_1}(q_2^1)$, is not reachable from the initial configuration $\mathcal{S}_0$ using only system rules. Namely, the signal c is never sent by any system rule, so A_1 never applies rule r_3, which is the only rule that gets the automaton A_1 to the "critical" state q_2^1.

However, in the presence of an intruder, a 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 the initial configuration $\mathcal{S}_0$ 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:

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

By inserting the signal c into the appropriate channel, intruder causes A_1 not to proceed within the hyper-cycle, but instead to apply the rule r_3 and break the hyper-cycle. The resulting configuration is critical, because 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 to an infinite trace with no hyper-cycles of $\mathcal{N}$:

$$
\begin{aligned}
&\longrightarrow_{r_4} Q_{A_1}(q_0^1),\ Q_{A_2}(q_2^2),\ Q_{A_3}(q_0^3),\\
&\quad R_{A_2,A_2}(*),\ R_{A_1,A_2}(*),\ R_{A_2,A_1}(*),\ R_{A_1,A_3}(\mathsf{a}),\\
&\quad R_{A_3,A_1}(*) = \mathcal{S}^1\\
&\longrightarrow_{p_1} Q_{A_1}(q_0^1),\ Q_{A_2}(q_2^2),\ Q_{A_3}(q_1^3),\\
&\quad R_{A_2,A_2}(*),\ R_{A_1,A_2}(*),\ R_{A_2,A_1}(*),\ R_{A_1,A_3}(*),\\
&\quad R_{A_3,A_1}(\mathsf{b}) = \mathcal{S}^2\\
&\longrightarrow_{p_2} Q_{A_1}(q_0^1),\ Q_{A_2}(q_2^2),\ Q_{A_3}(q_0^3),\\
&\quad R_{A_2,A_2}(*),\ R_{A_1,A_2}(*),\ R_{A_2,A_1}(*),\ R_{A_1,A_3}(*),\\
&\quad R_{A_3,A_1}(\mathsf{b}) = \mathcal{S}^3\\
&\longrightarrow_{r_5} Q_{A_1}(q_3^1),\ Q_{A_2}(q_2^2),\ Q_{A_3}(q_0^3),\\
&\quad R_{A_2,A_2}(*),\ R_{A_1,A_2}(*),\ R_{A_2,A_1}(*),\ R_{A_1,A_3}(*),\\
&\quad R_{A_3,A_1}(*) = \mathcal{S}^4\\
&\longrightarrow_{r_6} Q_{A_1}(q_0^1),\ Q_{A_2}(q_2^2),\ Q_{A_3}(q_0^3),\\
&\quad R_{A_2,A_2}(*),\ R_{A_1,A_2}(*),\ R_{A_2,A_1}(*),\ R_{A_1,A_3}(\mathsf{a}),\\
&\quad R_{A_3,A_1}(*) = \mathcal{S}^1\\
&\longrightarrow_{p_1} Q_{A_1}(q_0^1),\ Q_{A_2}(q_2^2),\ Q_{A_3}(q_1^3),\\
&\quad R_{A_2,A_2}(*),\ R_{A_1,A_2}(*),\ R_{A_2,A_1}(*),\ R_{A_1,A_3}(*),\\
&\quad R_{A_3,A_1}(\mathsf{b}) = \mathcal{S}^2\\
&\longrightarrow_{p_2} Q_{A_1}(q_0^1),\ Q_{A_2}(q_2^2),\ Q_{A_3}(q_0^3),\\
&\quad R_{A_2,A_2}(*),\ R_{A_1,A_2}(*),\ R_{A_2,A_1}(*),\ R_{A_1,A_3}(*),\\
&\quad R_{A_3,A_1}(\mathsf{b}) = \mathcal{S}^3\\
&\longrightarrow_{r_5} Q_{A_1}(q_3^1),\ Q_{A_2}(q_2^2),\ Q_{A_3}(q_0^3),\\
&\quad R_{A_2,A_2}(*),\ R_{A_1,A_2}(*),\ R_{A_2,A_1}(*),\ R_{A_1,A_3}(*),\\
&\quad R_{A_3,A_1}(*) = \mathcal{S}^4\\
&\cdots \longrightarrow_{r_6} \cdots \longrightarrow_{p_1} \cdots \longrightarrow_{p_2} \cdots \longrightarrow_{r_5} \cdots \longrightarrow_{r_6} \cdots
\end{aligned}
$$

(in a trace, rules p_2 *and* $r_5$can be permuted)

In this trace, automata A_1 and A_3 are playing an infinite game of ping-pong, while automaton A_2 is stuck. Note that none of the configurations $\mathcal{S}^1, \mathcal{S}^2, \mathcal{S}^3, \mathcal{S}^4$ is the initial configuration of $\mathcal{N}$, so there are no hyper-cycles of $\mathcal{N}$ in the above attack trace.

This example shows that even a well designed, functionally correct PAS, even a LAS, may no longer be periodic in the presence of an intruder. In the above example, it is sufficient for an intruder to perform only a single message insertion action to carry out the attack and change the periodic behavior of the system.

Security Complexity Results

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

Definition 6.4 (*Security Problem for Functionally Correct Systems*) The security problem restricted to functionally correct systems (SP-FCS) is defined as follows:

> **Input:** A functionally correct automata system $\mathcal{N}$, and an intruder $\mathcal{I}$.
> **Output:** Determine whether there is a successful attack, i.e., a trace of rules of $\mathcal{N}$ and $\mathcal{I}$ leading from the initial configuration of $\mathcal{N}$ into a critical configuration of $\mathcal{N}$.

Following Theorem 5.2, i.e., a PSPACE upper bound for the FCP for AS, we provide a PSPACE upper bound for the SP-FCS.

Theorem 6.5 (An upper bound for SP-FCS for AS) *The security problem for functionally correct systems for general* AS *and intruder belongs to PSPACE.*

Proof Let $\mathcal{N}$ be an AS and $\mathcal{I}$ an intruder. Recall that, since the number of channels and their capacity are fixed in advance, the total number of symbols contained in any system configuration is polynomial in $\mathcal{N}$ and $\mathcal{I}$.

Therefore, any trace that is representing an attack, i.e., a trace of $\mathcal{N}$ and $\mathcal{I}$ rules from the initial configuration to a critical configuration of $\mathcal{N}$, can be guessed and checked in NPSPACE [16]. Hence, checking the reachability of a critical configuration is in the NPSPACE complexity class. Since NPSPACE and PSPACE are the same complexity classes, and the FCP is in PSPACE, we conclude that the SP-FCS for general AS is in PSPACE. □

Theorem 6.6 (A lower bound for SP-FCS for LAS with an intruder using only one action) *The security problem for functionally correct systems is PSPACE-hard, even for*

functionally correct 1*-bounded* PAS *and the intruder that can apply at most one action.*

Proof (Proof Sketch) For the SP-FCS lower bound, to incorporate the intruder, we modify the proof of Theorem 5.3 accordingly.

The full proof is given below in the special Section "Proof of Theorem 6.6". □

The above complexity results are summarized in Table 1.

Detailed Proofs of Complexity Results

The following subsections contain detailed proofs of some of the complexitiy results. To help the reader, at the beginning of the subsection, before each proof, we restate the corresponding theorem.

Proof of Theorem 5.3

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

Remark 7.1 Definition 4.2 is dealing with rules in a general form:

$$\begin{array}{l} Q_A(\mathtt{q}),\ R_{B_1,A}(\mathtt{m}_1),\ldots,R_{B_k,A}(\mathtt{m}_k),\\ \quad R_{A,C_1}(*),\ldots,R_{A,C_\ell}(*)\ \longrightarrow\\ Q_A(\mathtt{q}'),\ R_{B_1,A}(*),\ldots,R_{B_k,A}(*),\\ \quad R_{A,C_1}(\mathtt{m}'_1),\ldots,R_{A,C_\ell}(\mathtt{m}'_\ell) \end{array} \tag{8}$$

where $\mathtt{m}_1,\ldots,\mathtt{m}_k,\mathtt{m}'_1,\ldots,\mathtt{m}'_\ell \in M_A$, and $\mathtt{q},\mathtt{q}' \in S_A$. The 'empty' channel $R_{A',A''}$ is represented as $R_{A',A''}(*)$.

For the sake of readability within our lower bounds, here, and henceforth, we will abbreviate the above (8) as

$$\mathtt{q}, R_{B_1,A}(\mathtt{m}_1),..,R_{B_k,A}(\mathtt{m}_k) \to \mathtt{q}', R_{A,C_1}(\mathtt{m}'_1),..,R_{A,C_\ell}(\mathtt{m}'_\ell) \tag{9}$$

"Being in the state $\mathtt{q}$, the automaton A consumes the event-driven signals m_1, m_2, ..., m_k, provided by B_1, B_2, ..., B_k, with generating the signals m'_1, m'_2, ..., m'_ℓ towards the intended recipients C'_1, C'_2, ..., C'_ℓ."

We also interpret the rule (9) in the operational way:

The automaton A can transform a precondition of the form:

$$R_{B_1,A}(\mathtt{m}_1),..,R_{B_k,A}(\mathtt{m}_k)$$

into a postcondition of the form:

$$R_{A,C_1}(\mathtt{m}'_1),..,R_{A,C_\ell}(\mathtt{m}'_\ell).$$

The PSPACE decision problem that we will simulate is 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.*"

For technical reasons, we reformulate the problem in terms of another Turing machine that we denote by $\widetilde{M}$, which deals only with one and the same initial configuration fixed in advance. Details are given in Lemma 7.2.

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 $\underbrace{aa..a}_{n\ times}$ 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):

$$\begin{cases} r_0 \longrightarrow r'_0,\ R_{A_0,A_1}(p) \\ R_{B_{n+1},A_0}(\widetilde{q}),\ r'_0, \longrightarrow r_0, \quad where\ \widetilde{q} \in \{\widetilde{q_0},\widetilde{q_1}\} \end{cases} \tag{10}$$

(a) 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}.
(b) Then A_0 is waiting for a signal $\widetilde{q}$ sent from B_{n+1} to end its hyper-cycle, with nullifying all channels.

We develop our AS by designing the automata we need step by step using a chain of lemmas. *To ease*

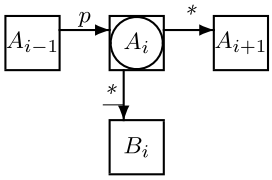
technicalities, we define the automata at hand only in terms of the tasks the automata should perform.

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).

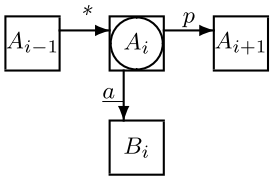
Providing $\widetilde{M}$'s Initial Tape

Lemma 7.3 *For $1 \le i \le n$, we design A_i so that A_i*

can transform a precondition of the form



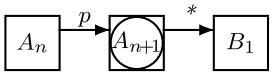
into a postcondition of the form



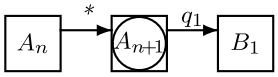
Then we provide the initial tape for $\widetilde{M}$, aa..a, by sequential execution of automata A_1, A_2, ..., A_n, resulting in the 'initial' non-empty channels $R_{A_1,B_1}(a)$, $R_{A_2,B_2}(a)$, ..., $R_{A_n,B_n}(a)$

Simulating $\widetilde{M}$'s Computations

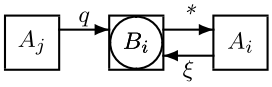
Lemma 7.4 *To provide the correct start of $\widetilde{M}$ with its initial state q_1, we design A_{n+1} so that A_{n+1} transforms the precondition produced by the nth step of Lemma 7.3*



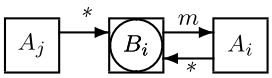
into a postcondition of the form



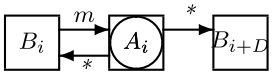
Lemma 7.5 *Given a Turing command $q\xi \to q'\eta D$, first we design B_i, $i = 1, .., n$, so that B_i can transform a precondition of the form, $j \neq i$,*



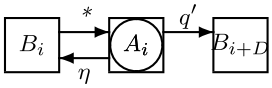
into the following postcondition, where m encodes the triple: $m = \langle q', \eta, D\rangle$:



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



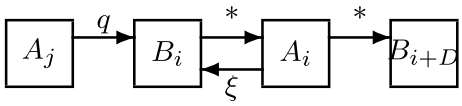
into the following postcondition,



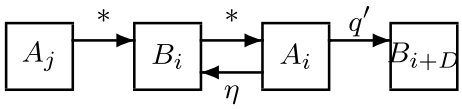
Lemma 7.6 *Any computation performed by $\widetilde{M}$ can be one-to-one simulated by running sequentially the corresponding ordered pairs of automata B_i and A_i.*

Proof Suppose that, being in state q and scanning ξ in ith 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 $\widetilde{q}' = \widetilde{q}$. □

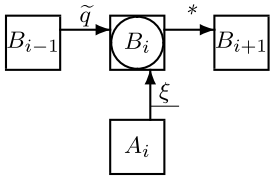
Definition 7.9 As critical we take system configurations which contain the fact $R_{A_2,B_1}(\widetilde{q_0})$.

Corollary 7.10 *The choice of critical configurations by Definition 7.9 provides PSPACE-hardness for functional correctness.*

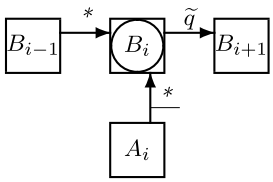
Proof Follows from Lemmas 7.2 and 7.8. □

Collecting Garbage

Lemma 7.11 *For $1 \le i \le n$, 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 7.11 applied sequentially.

Remark 7.12 Our system given in Section "Proof of Theorem 5.3" is a periodic system with a unique hyper-cycle from A_0 to A_0. □

Note that the periodic automata system developed in Section "Proof of Theorem 5.3 is functionally correct if and only if $\widetilde{M}$ does not terminate in $\widetilde{q_0}$, which is equivalent to the fact that there is no binary string x of length m accepted by M.

Recall that $R_{A_2,B_1}(\widetilde{q_0})$ is critical.

Bringing all the lemmas together, we can conclude that functional correctness is coPSPACE-hard even for periodic AS.

Proof of Theorem 5.7

Theorem FCP *for* 1*-bounded* PAS *is coNP-hard.*

For this result we encode the 3-SAT problem:

$$F(x_1, x_2, \ldots, x_n) = C_1 \wedge C_2 \wedge \cdots \wedge C_m \tag{11}$$

using a 1-bounded PAS, i.e., a LAS for which each of the automata instructions is applied exactly once in a hyper-cycle.

First, we introduce the automata $B_{0,0}$, $B_{0,1}$, $B_{0,2}$,..., $B_{0,n}$, to non-deterministically assign x_j the value α_j, that is t_j or f_j, where t_j encodes "x_j is true", and f_j encodes "x_j is false".

Given α_1, ..., α_n, and a disjunct C_ℓ, we will calculate the corresponding value $C_\ell(\alpha_1, \alpha_2, \dots, \alpha_n)$ by means of a group of automata $B_{\ell,0}$, $B_{\ell,1}$, $B_{\ell,2}$, ..., $B_{\ell,n}$. See Lemma 7.15.

Definition 7.13 We use 'leading' signals c_0, d_0, c_1, d_1, κ_1,..., c_ℓ, d_ℓ, κ_ℓ,..., to control the execution steps.

(a) The 'main controller' $B_{0,0}$ specified by the following instructions (r_{00} is its initial state) starts a hyper-cycle of the entire system:

$$\begin{cases} r_{00} \longrightarrow r'_{00},\ R_{B_{0,0},B_{0,1}}(d_0) \\ R_{B_{m,n},B_{0,0}}(d_m),\ r'_{00} \longrightarrow r_{00} \\ R_{B_{m,n},B_{0,0}}(c_m),\ r'_{00} \longrightarrow r_{00} \\ R_{B_{m,n},B_{0,0}}(\kappa_m),\ r'_{00} \longrightarrow r_{00} \end{cases} \tag{12}$$

(b) For $1 \le j < n$, a non-deterministic $B_{0,j}$ is specified by the instructions ($r_{0,j}$ is its initial state):

$$\begin{cases} r_{0,j},\ R_{B_{0,j-1},B_{0,j}}(d_0) \longrightarrow r_{0,j},\ R_{B_{0,j},B_{0,j+1}}(d_0),\ R_{B_{0,j},B_{1,j}}(t_j) \\ (\texttt{take}\ t_j) \\ r_{0,j},\ R_{B_{0,j-1},B_{0,j}}(d_0) \longrightarrow r_{0,j},\ R_{B_{0,j},B_{0,j+1}}(d_0),\ R_{B_{0,j},B_{1,j}}(f_j) \\ (\texttt{take}\ f_j) \end{cases} \tag{13}$$

(c) For $j = n$, $B_{0,j}$ is specified by the instructions ($r_{0,j}$ is its initial state):

$$\begin{cases} r_{0,j},\ R_{B_{0,j-1},B_{0,j}}(d_0) \longrightarrow r_{0,j},\ R_{B_{0,j},B_{1,0}}(d_0),\ R_{B_{0,j},B_{1,j}}(t_j) \\ (\texttt{take}\ t_j) \\ r_{0,j},\ R_{B_{0,j-1},B_{0,j}}(d_0) \longrightarrow r_{0,j},\ R_{B_{0,j},B_{1,0}}(d_0),\ R_{B_{0,j},B_{1,j}}(f_j) \\ (\texttt{take}\ f_j) \\ r_{0,j},\ R_{B_{0,j-1},B_{0,j}}(c_0) \longrightarrow r_{0,j},\ R_{B_{0,j},B_{1,0}}(d_0),\ R_{B_{0,j},B_{1,j}}(f_j) \\ (\texttt{usedlater}) \end{cases} \tag{14}$$

(d) For $1 \le \ell \le m$, $B_{\ell,0}$ is specified by the instructions ($r_{\ell,0}$ is its initial state):

$$\begin{cases} r_{\ell,0},\ R_{B_{\ell-1,n},B_{\ell,0}}(d_{\ell-1}) \longrightarrow r_{\ell,0},\ R_{B_{\ell,0},B_{\ell,1}}(c_\ell) & [\texttt{start}\ B_{\ell,1}] \\ r_{\ell,0},\ R_{B_{\ell-1,n},B_{\ell,0}}(c_{\ell-1}) \longrightarrow r_{\ell,0},\ R_{B_{\ell,0},B_{\ell,1}}(\kappa_\ell) & [C_{\ell-1}\ \texttt{notvalid}] \\ r_{\ell,0},\ R_{B_{\ell-1,n},B_{\ell,0}}(\kappa_{\ell-1}) \longrightarrow r_{\ell,0},\ R_{B_{\ell,0},B_{\ell,1}}(\kappa_\ell) & [(11)\ \textit{notvalid}] \end{cases} \tag{15}$$

(e) For $1 \le j \le n$ and $1 \le \ell < m$, $B_{\ell,j}$ is specified as ($s_{\ell,j}$ is its initial state):

$$\begin{cases} s_{\ell,j},\ R_{B_{\ell,j-1},B_{\ell,j}}(c_\ell),\ R_{B_{\ell-1,j},B_{\ell,j}}(\xi_j) \longrightarrow \\ \quad s_{\ell,j},\ R_{B_{\ell,j},B_{\ell,j+1}}(d_\ell),\ R_{B_{\ell,j},B_{\ell+1,j}}(\xi_j) & \text{if } \xi_j \text{ makes} C_\ell \text{ valid} \\ s_{\ell,j},\ R_{B_{\ell,j-1},B_{\ell,j}}(c_\ell),\ R_{B_{\ell-1,j},B_{\ell,j}}(\xi_j) \longrightarrow \\ \quad s_{\ell,j},\ R_{B_{\ell,j},B_{\ell,j+1}}(c_\ell),\ R_{B_{\ell,j},B_{\ell+1,j}}(\xi_j) & \text{otherwise} \\ s_{\ell,j},\ R_{B_{\ell,j-1},B_{\ell,j}}(d_\ell),\ R_{B_{\ell-1,j},B_{\ell,j}}(\xi_j) \longrightarrow \\ \quad s_{\ell,j},\ R_{B_{\ell,j},B_{\ell,j+1}}(d_\ell),\ R_{B_{\ell,j},B_{\ell+1,j}}(\xi_j) \\ s_{\ell,j},\ R_{B_{\ell,j-1},B_{\ell,j}}(\kappa_\ell),\ R_{B_{\ell-1,j},B_{\ell,j}}(\xi_j) \longrightarrow \\ \quad s_{\ell,j},\ R_{B_{\ell,j},B_{\ell,j+1}}(\kappa_\ell),\ R_{B_{\ell,j},B_{\ell+1,j}}(\xi_j) \end{cases} \tag{16}$$

Here ξ_j is t_j or f_j. For $j = n$, we use $B_{\ell,j+1}$ above as a nickname for $B_{\ell+1,0}$.

(f) For $1 \le j \le n$ and $\ell = m$, $B_{\ell,j}$ is specified as ($s_{\ell,j}$ is its initial state):

$$\begin{cases} s_{\ell,j},\ R_{B_{\ell,j-1},B_{\ell,j}}(c_\ell),\ R_{B_{\ell-1,j},B_{\ell,j}}(\xi_j) \longrightarrow s_{\ell,j},\ R_{B_{\ell,j},B_{\ell,j+1}}(d_\ell), \\ \quad \text{if } \xi_j \text{makes} C_\ell \text{valid} \\ s_{\ell,j},\ R_{B_{\ell,j-1},B_{\ell,j}}(c_\ell),\ R_{B_{\ell-1,j},B_{\ell,j}}(\xi_j) \longrightarrow s_{\ell,j},\ R_{B_{\ell,j},B_{\ell,j+1}}(c_\ell), \\ \quad \text{otherwise} \\ s_{\ell,j},\ R_{B_{\ell,j-1},B_{\ell,j}}(d_\ell),\ R_{B_{\ell-1,j},B_{\ell,j}}(\xi_j) \longrightarrow s_{\ell,j},\ R_{B_{\ell,j},B_{\ell,j+1}}(d_\ell), \\ s_{\ell,j},\ R_{B_{\ell,j-1},B_{\ell,j}}(\kappa_\ell),\ R_{B_{\ell-1,j},B_{\ell,j}}(\xi_j) \longrightarrow s_{\ell,j},\ R_{B_{\ell,j},B_{\ell,j+1}}(\kappa_\ell), \end{cases} \tag{17}$$

Here ξ_j is t_j or f_j. For $j = n$, we use $B_{\ell,j+1}$ above as a nickname for $B_{0,0}$.

We conclude the proof of Theorem 5.7 with the following lemmas on a polynomial-time reduction and the size of its input.

Lemma 7.14 *There is a polynomial p such that the size of the automata system in Definition 7.13 is bounded by $p(nm)$.*

Lemma 7.15 *The system given in Definition 7.13 is a 1-bounded periodic system.*

Moreover, an SAT instance (11) is satisfiable iff there is a sequence of actions in at least one hyper-cycle starting with $B_{0,0}$ and leading to a critical configuration of the system - that is the configurations in which $R_{B_{m,n},B_{0,0}}(d_m)$ is observed.

Proof Due to the 'leading' signals d_0, c_1, d_1, κ_1,..., c_ℓ, d_ℓ, κ_ℓ ,..., any hyper-cycle consists in sequential execution of the automata (each runs only once):

$$\begin{array}{l} B_{0,0}, B_{0,1}, B_{0,2}, \dots, B_{0,n}, \\ B_{1,0}, B_{1,1}, B_{1,2}, \dots, B_{1,n}, \\ B_{2,0}, B_{2,1}, B_{2,2}, \dots, B_{2,n}, \ \dots, \\ B_{\ell,0}, B_{\ell,1}, B_{\ell,2}, \dots, B_{\ell,n}, \ \dots, \\ B_{m,0}, B_{m,1}, B_{m,2}, \dots, B_{m,n}, B_{0,0} \end{array}$$

According to (13), $B_{0,0}$, $B_{0,1}$, $B_{0,2}$,..., $B_{0,n}$, provides non-deterministically the values α_j, that is t_j or f_j.

According to (16), $B_{1,0}$, $B_{1,1}$, $B_{1,2}$,..., $B_{1,n}$, first provides the transit of these α_j to the next level by consuming α_j from $R_{B_{0,j},B_{1,j}}$ ($R_{B_{0,j},B_{1,j}}$ becomes empty) and then writing α_j into the corresponding $R_{B_{1,j},B_{2,j}}$ on the next level. Second, $B_{1,0}$ changes d_0 in c_1, see (15).

Given C_1 to be examined, we consider two cases.

Case 1. $C_1(\alpha_1, \dots, \alpha_n)$ is valid. Then by consecutive examining pairs C_1, α_j, (16) provides the final "positive" $R_{B_{1,n},B_{2,0}}(d_1)$.

Case 2. $C_1(\alpha_1, \dots, \alpha_n)$ is not valid. Then by consecutive examining pairs C_1, α_j (16) provides the final "negative" $R_{B_{1,n},B_{2,0}}(c_1)$, since on the next level $B_{2,0}$ changes c_1 into killing κ_2, see (15).

Similarly, on any level ℓ, by consecutive examining pairs C_ℓ, α_j, (16) provides either the final "positive" $R_{B_{\ell,n},B_{\ell+1,0}}(d_\ell)$ or one of the final "negatives" $R_{B_{\ell,n},B_{\ell+1,0}}(c_\ell)$ and $R_{B_{\ell,n},B_{\ell+1,0}}(\kappa_\ell)$. □

This concludes the proof of Theorem 5.7.

Remark 7.16 The AS used in the encoding is a 1-bounded PAS, since each of the automata instructions is applied no more than once in a hyper-cycle. The first application of an automaton nullifies all its incoming channels.

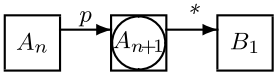
Proof of Theorem 6.6

Theorem SP-FCS *problem is PSPACE-hard, even for functionally correct* 1*-bounded* PAS *and the intruder that can apply at most one action.*

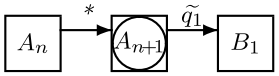
As input to the problem we take an AS from Definition 7.17 and an intruder from Definition 7.20.

Definition 7.17 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 7.18.

Lemma 7.18 *We update A_{n+1} so that A_{n+1} can transform the precondition produced by the n th figm 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 7.19 *The system given in Definition 7.17 is a* 1*-bounded* PAS *which is functionally correct.*

Proof According to Lemmas 7.3, 7.18, and 7.11, we can develop a unique hyper-cycle from A_0 to A_0 by running sequentially the following automata:

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

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 $A_1, \dots, A_n$ results in the non-empty channels $R_{A_1,B_1}(a)$, ..., $R_{A_n,B_n}(a)$. At once Lemma 7.18 redirects the execution to the garbage collecting Lemma 7.11, 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. That is, the AS used in the encoding is a 1-bounded PAS.

Definition 7.20 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}:

$$R_{A_{n+1},B_1}(\widetilde{q_1}) \longrightarrow R_{A_{n+1},B_1}(q_1) \tag{18}$$

Lemma 7.21 *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 7.17 enriched with the intruder action* (18).

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 7.18, 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 6.6.

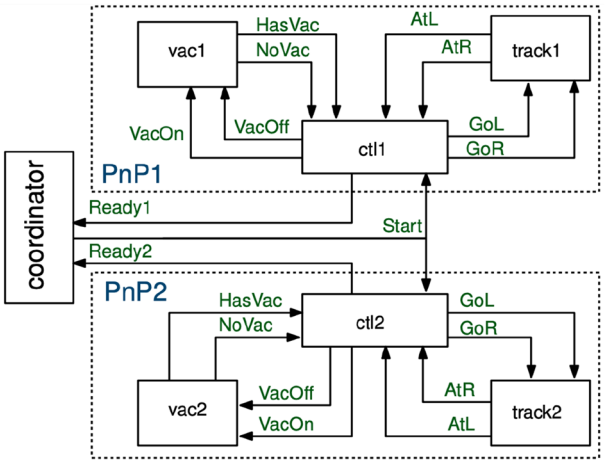


Fig. 4 2PnP Function Blocks, with event labelled connections and a coordinator [20]

Corollary 7.22 *The security problem for functionally correct systems given in Definition 6.4 is PSPACE-complete, even in the case the intruder in question can apply only one action.*

These security problems are still PSPACE-complete even in the case of a PAS *and a* 1*-bounded* PAS*, and even in the case the intruder can apply only one action.*

Proof The statements follow from Theorems 6.5 and 6.6. □

Automated Verification

The formal models, verification problems, and complexity results presented here provide foundations for automated security verification of I4.0 applications. To illustrate this, we summarize experiments conducted using the Maude formalization described in [19, 20].

Our main goal is to evaluate that these periodic systems are suitable for automated verification using proof-of-concept scenarios based on the PnP described in Section "Motivating Example".

Different scenarios used in the experiments are listed below. In these experiments, we investigated the effects of varying the intruder bound and increasing the size of the application.

Different types of PnP scenarios

PnP- This is the scenario described in Section "Motivating Example".

2PnP- This scenario is a LAS that contains two instances of PnP and a coordinator that ensures that each instance of PnP starts at the same time, which is at the beginning of the hyper-cycle. Figure 4 shows the function blocks for this scenario and their connections.

PnP-2Msgs—This scenario modifies the logic of PnP so that when the track is on the right (to pick up caps), it waits for two signals before moving to the left (to place a cap): GoL from ctl and HasVac/NoVac from vac; and if vac is on, two signals are required to turn it off: VacOff from ctl and atL from track. The point is to force the intruder to perform at least two intrusions to bring the modified system into a critical configuration.

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

Table 2 Model-checking results for the SP-FCS using Maude for different scenarios[3, Table 2]

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 (×3.7)	40	no
	1	388 (×4.6)	182	yes
	2	2873 (×7.1)	1409	yes
	3	26440 (×16.0)	19631	yes
PnP-2Msgs	0	29 (×1.3)	40	no
	1	722 (×8.5)	177	no
	2	1854 (×4.6)	912	yes
	3	10248 (×6.2)	4965	yes
2PnP-2Msgs	0	114 (×4.9)	88	no
	1	6814 (×81.1)	5277	no
	2	22179 (×54.1)	18208	yes
	3	153824 (×93.1)	225898	yes

The values in parentheses, ×*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

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

Maude I4.0 Formal Model

In the following we give a brief overview of the Maude [6] rewriting logic model of I4.0 applications illustrated by PnP. Such models represent the system state as terms of an order-sorted algebra. The system behavior is specified by local rewrite rules that describe how the system state changes over time. A more detailed description of the Maude implementation can be found in our previous work [19] and the code with the experiments can be found in https://github.com/SRI-CSL/WrapPat.

As illustrated in Section "Motivating Example", an I4.0 application consists of a set of interconnected interactive finite state machines called function blocks (FB). For theoretical analysis, in this paper FBs are represented as automata systems. The Maude representation of an FB is a term of the form `[fbId : fbCid | fbAttrs]`, where

- `fbId` is the FB identifier,
- `fbCid` is its class identifier, and
- `fbAttrs` is a set of attribute-value pairs, including `(state : st)`, `(oEvEffs : oeffs)`, and `(ticked : b)`, with
  - `state`, `oEvEffs`, `ticked` being the attribute tags,
  - `st` the current state,
  - `oeffs` a set of signals/events to be transmitted (out effects), and
  - `b` a boolean indicating whether the FB has fired a transition in the current cycle.

An FB transition is a term of the form `tr(st0,st1,cond,oeffs)` where `st0` is the initial state and `st1` is the final state, `cond` is the condition, and `oeffs` is the set of outputs. A condition is a boolean combination of primitive conditions `(in is ev)` that specifies a particular event (`ev`) at input `in`. For example, the condition `inEv("VacOff") is ev("VacOff")` says that the event `ev("VacOff")` must be on input `inEv("VacOff")`. The elements of `oeffs` are of the form `(out : ev)`, which specifies the event `ev` on the output `out`. An example of the `oeff` is `(outEv("NoVac") : ev("NoVac"))`. The transition `tr(st0,st1,cond,oeffs)` is enabled by a set of inputs if they satisfy `cond` and the current state of the function block state `st0`. In this case, the transition can fire, changing the function block state to `st1` and adding `oeffs` to the `oEvEffs` attribute.

Example: a vacuum FB

A vacuum FB (class id `vac`) has states

```
st("off"), st("on"), st("on-novac");
```

and outputs

```
inEv("VacOn"), inEv("VacOff");
```

The initial state, `vacInit(id("vac"))`, of a vacuum FB with identifier `id("vac")` is defined by

```
id("vac") is defined by
  vacInit(id("vac")) =
   [id("vac") : vac | state : st("off") ; ticked : false ;
                      iEvEffs : none ; oEvEffs : none]
```

The vac FB class has 3 transitions: turn the vacuum off, if it is on and there is an"off" message; or turn the vacuum on, if it is off and there is an "on" message. In the latter case, there are two FB transitions, one for the case that the result is a vacuum, and one for the case that no vacuum is obtained, for example, because the cap to be picked up is slightly off position and there is an air gap.

```
tr(st("on"), st("off"), inEv("VacOff") is ev("VacOff"),
   outEv("NoVac") :~ ev("NoVac"))
tr(st("off"), st("on-novac"), inEv("VacOn") is ev("VacOn"),
   outEv("NoVac") :~ ev("NoVac"))
tr(st("off"), st("on"), inEv("VacOn") is ev("VacOn"),
   outEv("HasVac") :~ ev("HasVac"))
```

The initial state of the PickNPlace (PnP) application described in Section "Motivating Example" is

```
[id("pnp") | fbs : (ctlInit(id("ctl")
                trackInit(id("track")) vacInit(id("vac"))) ;
  iEMsgs : {{id("ctl"),inEv("start")},ev("start")} ;
  oEMsgs : none ; ssbs : none]
```

where the message `|id("ctl"),inEv("start"),ev("start")|` starts the application controller.

The flow of information, depicted as labeled arrows in Figs. 1 and 4, is formally represented as terms of the sort `Link` A link connecting output ports of one FB (`fbId0`) to the inputs of another (possibly the same) FB (`fbId1`) has the form `|fbId0,out,fbId1,in|`.

The following two links connect the vac outputs to the controller (ctl) inputs.

```
{{id("vac"),outEv("NoVac")}, {id("ctl"),inEv("NoVac")}}
{{id("vac"),outEv("HasVac")},{id("ctl"),inEv("HasVac")}}
```

Application Execution Rules There are two rewrite rules that specify the behavior of the application, [app-exe1] and [app-exe2], and a rule that models bounded intruder actions [app-intruder]. The rule [app-exe1] fires an enabled FB transition for each FB with ticked attribute false. The ticked attribute of that FB is set to true.

When [app-exe1] is no longer applicable, [app-exe2] fires. This collects the ouputs from each FB oEvEffs attribute, uses the application links to route them to the target FB iEvEffs attributes, and prepares for the next cycle by resetting the FB ticked attributes to false. The use of the ticked attribute of an FB ensures that an FB fires at most one transition per cycle. The [app-intruder] can fire at any point, if there is a message remaining in the set of intruder messages. In this case, a message is removed from the set of intruder messages and added to the system messages to be delivered.

We illustrate the initial steps of an execution of the PNP app using the two types of rewrite rules. Consider the following initial configuration, initPNP, of the PNP in which the system is ready to start. This is specified by the controller's ctl state being init, and by the start message to be delivered to the controller. This message is generated, for example, when a user presses a start button in the factory element.

```
[id("pnp") | fbs :
 ([id("ctl") : ctl | state : st("init") ;
  oEvEffs : none ; ticked : false]
  [id("track") : track | state : st("L") ;
  oEvEffs : none ; ticked : false]
  [id("vac") : vac | state : st("off") ;
  oEvEffs : none ; ticked : false]) ;
iEMsgs : {{id("ctl"),inEv("start")},ev("start")} ;
oEMsgs : none ;  ssbs : none]
```

Applying [app-exe1] the start message for ctl is delivered using the ctl transition

```
tr(st("init"), st("LOff"),
   inEv("start") is ev("start"),
   outEv("GoR") :~ ev("GoR"))
```

Now ctl is in state LOff and has a pending output.

```
[id("pnp") | fbs : (
  [id("ctl") : ctl | state : st("LOff") ;
   oEvEffs : (outEv("GoR") :~ ev("GoR")) ; ticked : true]
  [id("track") : track | state : st("L") ;
   oEvEffs : none ; ticked : false]
  [id("vac") : vac | state : st("off") ;
   oEvEffs : none ; ticked : false] ) ;
iEMsgs : none ; oEMsgs : none ; ssbs : none]
```

There are no more possible transitions, i.e., no more solutions, so the next rewrite applies [app-exe2]. (outEv("GoR") : ev("GoR")) is collected from ctl's oEvEff attribute and is routed to track using the link:

id(εctlε), outEv(εGoRε), id(εtrackε), inEv(εGoRε)

```
          id("ctl"),outEv("GoR"), id("track"), inEv("GoR")
[id("pnp") | fbs : (
  [id("ctl") : ctl | state : st("LOff") ;
   oEvEffs : none ; ticked : false]
  [id("track") : track | state : st("L") ;
   oEvEffs : none ; ticked : false]
  [id("vac") : vac | state : st("off") ;
   oEvEffs : none ; ticked : false]) ;
 iEMsgs : {{id("track"),inEv("GoR")},ev("GoR")} ;
 oEMsgs : none ; ssbs : none]
```

Now rule [app-exe1] can be applied to fire the track transition

```
tr(st("L"), st("mvR"),
  inEv("GoR") is ev("GoR"),
  outEv("GoR1") :~ ev("GoR1"))
```

This produces a configuration with a pending message from `track`.

```
[id("pnp") | fbs : (
  [id("ctl") : ctl | state : st("LOff") ;
    oEvEffs : none ; ticked : false]
  [id("track") : track | state : st("mvR") ;
   oEvEffs : (outEv("GoR1") :~ ev("GoR1")) ; ticked : true]
  [id("vac") : vac | state : st("off") ;
  oEvEffs : none ; ticked : false] ) ;
  iEMsgs : none ; oEMsgs : none ; ssbs : none]
```

Given the specification of these rules, we can use Maude's built-in search engine to check for reachability. For example, the command

```
search[1] initPNP =>* app such that critical(app) .
```

is used to search whether a critical configuration `app`, specified by the function `critical`, can be reached from the initial configuration `initPNP`.

The PickNPlace experiments show that it is feasible in practice to formally verify simple realistic scenarios. 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. The search space also increases as the bound on allowed intruder interventions increases. In particular, in the experiment summarized in Table 2, doubling the application size approximately quadruples the number of states that need to be explored to verify functional correctness.

Increasing the attacker bound leads to further significant increase in the number of states explored to find an attack, but the amount of increase depends on where in the search space an attack is found. Thus, the increase in searched state space is not a uniform function of system size and bound on attacker resources.

As the last column in Table 2 shows, all modeled scenarios are functionally correct. It also shows that in the PnP-2Msgs and 2PnP-2Msgs scenarios, the intruder needs at least two actions to carry out an attack. This explains the larger factor in the bound of 1 message rows in the *Number of Configurations Explored* column, since the entire state space must be explored to determine that there are no attacks.

Although the analyses are carried out on abstract models, in [19, 20] we show how the abstract model can be automatically transformed into a model of a deployed system in which automata functionality is executed on multiple devices connected over a network. The results of the analyses of the abstract models can be lifted to deployed system models, since the transformations preserve the safety and security properties of the abstract model.

The Maude code along with documentation, scenarios and sample runs can be found at https://github.com/SRI-CSL/WrapPat.

Conclusions and Future Work

In this paper we present a formal model called *automata systems*. The model is based on multiset rewriting and was developed for the specification and verification of automated interconnected systems, such as I4.0 applications. Motivated by the properties of concrete I4.0 applications, such as periodicity, we identified several classes of systems within the general AS framework. Each class represents the specific characteristics of interest that represent the requirements of I4.0 applications.

Due to the nature of communication between insecure devices within the system, opportunities for cyber-attacks arise. To analyze such security issues, we also present a range of intruder models. We vary the strength of the intruder, and consider intruders for which the number of intrusions into the system is restricted.

Several related safety and security verification problems are defined and investigated. We obtain a comprehensive collection of complexity results, including several security complexity results involving different types of intruders.

We also demonstrate that our formal model is suitable for automated verification. We describe an executable specification of periodic AS in the rewriting tool Maude. We have conducted several experiments with I4.0 specifications using proof-of-concept examples. In particular, these experiments can help understand the security level of applications under different assumptions about the strength of the intruder.

There are a number of ways in which the model of automata systems presented in this paper can be extended. For example, systems with smart devices as components, systems with distributed manufacturing, and similar systems in which communication between system components takes place via cryptographic network protocols can be considered. At the same time, an intruder model more similar to the DY intruder could be modelled by adding encryption capabilities.

Furthermore, it may be useful to consider timed systems and intruder models that take into account physical properties, such as distances and processing time. For such extensions to the formal AS model, some of the approaches from our earlier work [14, 15] could be adapted. In addition, systems and intruders with different resource-sensitive features can lead to investigations of other verification problems, such as those considered in [22, 23]. Providing the relevant details and the necessary expressiveness would allow us to avoid some false positives.

Statistical model-checking provides another avenue for analysing the security of I4.0 applications. Techniques similar to the work in [2], could be applied to study different intruder and defence strategies and their success rates.

We also intend to investigate abstraction techniques and properties that are similar to those considered in [20] relating to different extensions of our model.

Finally, while the experiments provided with automated verification tools are promising, we plan to investigate possible optimizations to improve performance.

Acknowledgements 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 was partially supported by the U. S. Office of Naval Research under award number N00014-20-1-2635. Talcott was partially supported by the U. S. Office of Naval Research under award numbers N00014-15-1-2202 and N00014-20-1-2644, and NRL grant N0017317-1-G002.

Funding Funding support the authors received is as given in the **Acknowledgements** paragraph above.

Declaration

Conflict of interest The authors declare that they have no conflicting or competing interests.

Code availibility Code developed for this work is available at https://github.com/SRI-CSL/WrapPat.

References

1. Ademaj et al. Time sensitive networks for flexible manufacturing testbed—description of converged traffic types, IIC white paper 2019.
2. AlTurki MA, Kanovich M, Ban Kirigin T, Nigam V, Scedrov A, Talcott C. 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, 60–71. ACM 2018. https://dl.acm.org/doi/10.1145/3264888.3264895
3. AlTurki MA, Ban Kirigin T, Kanovich M, Nigam V, Scedrov A, Talcott C. On security analysis of periodic systems: expressiveness and complexity. In: ICISSP 2021-Proceedings of the 7th International Conference on information systems security and privacy. 2021;1:43–54.
4. Biere A, Cimatti A, Clarke EM, Strichman O, Zhu Y. Bounded model checking. Adv Comput. 2003;58:117–48.
5. Cyberattack has caused confirmed physical damage for the second time ever. 2015. https://www.wired.com/2015/01/german-steel-mill-hack-destruction/. Accessed 30 Sep 2021
6. Clavel M, Durán F, Eker S, Lincoln P, Martí-Oliet N, Meseguer J, Talcott C. All about Maude: a high-performance logical framework, volume 4350 of LNCS. Berlin: Springer; 2007.
7. Dolev D, Yao A. On the security of public key protocols. IEEE Trans Inf Theory. 1983;29(2):198–208.
8. Durgin NA, Lincoln P, Mitchell JC, Scedrov A. Multiset rewriting and the complexity of bounded security protocols. J Comput Secur. 2004;12(2):247–311.
9. Enderton HB. A mathematical introduction to logic. Cambridge: Academic Press; 1972.
10. ENISA. Good practices for security of internet of things in the context of smart manufacturing 2018. https://www.enisa.europa.eu/publications/good-practices-for-security-of-iot. Accessed 30 Sep 2021
11. Fiat M, et al. OPC UA security analysis 2017. https://opcfoundation.org/wp-content/uploads/2017/04/OPC_UA_security_analysis-OPC-F-Responses-2017_04_21.pdf. Accessed 30 Sep 2021
12. Kanovich M, Ban Kirigin T, Nigam V, Scedrov A. Bounded memory protocols and progressing collaborative systems. In: Crampton J, Jajodia S, Mayes K, editors. Computer Security—ESORICS. 2013;2013:309–26.
13. Kanovich MI, Ban Kirigin T, Nigam V, Scedrov A. Bounded memory Dolev-Yao adversaries in collaborative systems. Inf Comput. 2014;238:233–61.
14. Kanovich MI, Ban Kirigin T, Nigam V, Scedrov A, Talcott CL, Perovic R. A rewriting framework and logic for activities subject to regulations. Math Struct Comput Sci. 2017;27(3):332–75.
15. Kanovich MI, Ban Kirigin T, Nigam V, Scedrov A, Talcott CL. Time, computational complexity, and probability in the analysis of distance-bounding protocols. J Comput Secur. 2017;25(6):585–630.
16. Kanovich MI, Rowe P, Scedrov A. Collaborative planning with confidentiality. J Autom Reason. 2011;46(3–4):389–421.
17. Lanotte R, Merro M, Munteanu A, Viganò L. A formal approach to physics-based attacks in cyber-physical systems. ACM Trans Priv Secur. 2020;23(1):1–41. https://dl.acm.org/doi/10.1145/3373270
18. Leveson NG, Thomas JP. STPA handbook. 2018. https://psas.scripts.mit.edu/home/get_file.php?name=STPA_handbook.pdf. Accessed 30 Sep 2021
19. Nigam V, Talcott C. Formal security verification of industry 4.0 applications. In: The 24th IEEE International Conference on emerging technologies and factory automation (ETFA), special track on cybersecurity in industrial control systems, 2019;1043–1050. https://ieeexplore.ieee.org/document/8869428
20. Nigam V, Talcott C. Automated construction of security integrity wrappers for Industry 4.0 applications. In: The 13th International Workshop on rewriting logic and its applications, volume 12328 of *LNCS*, 2020; p. 197–215.
21. Savage JE. Models of computation. Reading: Addison-Wesley; 1998.
22. Urquiza AA, AlTurki MA, Kanovich M, Ban Kirigin T, Nigam V, Scedrov A, Talcott C. Resource and timing aspects of security protocols. J Comput Secur. 2021;29(3):299–340.
23. Urquiza AA, AlTurki MA, Kanovich M, Ban Kirigin T, Nigam V, Scedrov A, Talcott C . Resource-bounded intruders in denial of service attacks. In: 32nd Computer Security Foundations Symposium (CSF), 2019; p. 382–96. IEEE.
24. Yoong LH, Roop PS, Bhatti ZE, Kupz MMY. Model-driven design using IEC 61499: a synchronous approach for embedded automation systems. Berlin: Springer; 2015.
25. Zoitl A, Lewis R. Modelling control systems using IEC 61499. In: Control Engineering Series 95. London: The Institution of Electrical Engineers; 2014. https://www.amazon.com/Modelling-Control-Systems-Robotics-Sensors/dp/1849197601