Runtime Enforcement for Control System Security

Ruggero Lanotte
University of Insubria
Como, Italy
ruggero.lanotte@uninsubria.it

Massimo Merro
University of Verona
Verona, Italy
massimo.merro@univr.it

Andrei Munteanu
University of Verona
Verona, Italy
andrei.munteanu@univr.it

***Abstract*—With the explosion of *Industry 4.0*, industrial facilities and critical infrastructures are transforming into "smart" systems that dynamically adapt to external events. The result is an ecosystem of heterogeneous physical and cyber components, such as *programmable logic controllers*, which are more and more exposed to *cyber-physical attacks*, *i.e.*, security breaches in cyberspace that adversely affect the physical processes at the core of *industrial control systems*.**

We apply *runtime enforcement* techniques, based on an ad-hoc sub-class of Ligatti et al.'s *edit automata*, to enforce specification compliance in networks of potentially compromised controllers, formalised in Hennessy and Regan's *Timed Process Language*. We define a synthesis algorithm that, given an alphabet $\mathcal{P}$ of observable actions and an enforceable regular expression e capturing a timed property for controllers, returns a monitor that enforces the property e during the execution of any (potentially corrupted) controller with alphabet $\mathcal{P}$ and complying with the property e. Our monitors *correct* and *suppress* incorrect actions coming from corrupted controllers and *emit* actions in full autonomy when the controller under scrutiny is not able to do so in a correct manner. Besides classical properties, such as *transparency* and *soundness*, the proposed enforcement ensures non-obvious properties, such as *polynomial complexity* of the synthesis, *deadlock- and divergence-freedom of monitored controllers*, together with *scalability* when dealing with networks of controllers.

***Index Terms*—Runtime enforcement, process calculus, control system security, PLC malware**

I. Introduction

Industrial Control Systems (ICSs) are integrations of networking and distributed computing systems with physical processes, where feedback loops allow the latter to affect the computations of the former and vice versa. Historically, ICSs relied on proprietary technologies and were implemented as standalone networks in physically protected locations. However, with the introduction of *Smart Manufacturing* (*Industry 4.0*) the growing connectivity and integration of these systems has triggered a dramatic increase in the number of *cyber-physical attacks* [1], *i.e.*, security breaches in cyberspace that adversely affect the physical processes. Some notorious examples are: (i) the *Stuxnet* worm, which reprogrammed PLCs of nuclear centrifuges in the nuclear facility of Natanz in Iran [2]; (ii) the *BlackEnergy* cyber-attack on the Ukrainian power grid [3]; (iii) the recent *Triton* malware that targeted a petrochemical plant in Saudi Arabia [4]. The gravity of such attacks has been addressed in high-level forums such as the 2018 World Economic Forum meeting in Davos.

Published scan data shows how thousands of ICS components, and in particular *programmable logic controllers* (PLCs), are directly accessible from the Internet to improve efficiency [5], [6]. Furthermore, controllers are often connected to each other in so called *field communications networks*, opening the way to the spreading of ad-hoc worms coming from few infected ones (see Figure 1).

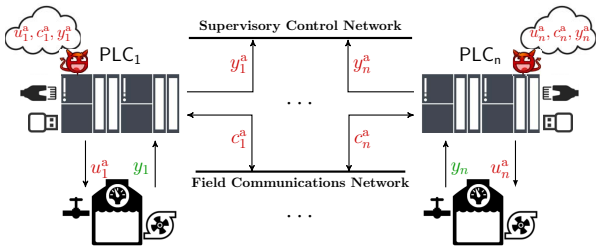

Fig. 1. A network of compromised controllers: y_i denote genuine sensor readings from the plant, y_i^{a} corrupted sensor readings sent from the PLC, u_i^{a} corrupted actuator commands, and c_i^{a} corrupted inter-controller communication channels.

Programmable Logic Controllers have an ad-hoc architecture to execute simple repeating processes known as *scan cycles*. Each scan cycle consists of three steps: (i) reading of *sensor measurements* of the physical process; (ii) execution of the controller code to determine how the physical process should change according to both sensor measurements and potential interactions with other controllers; (iii) transmission of *commands* to the *actuator devices* to implement the calculated changes. The scan cycle of a controller must be completed within a specific time, called *maximum cycle limit*, which depends on the controlled physical process; if this time limit is violated the controller stops and throws an exception [7].

Due to their sensitive role in controlling industrial processes, successful exploitation of a controller can have severe consequences on ICSs. In fact, although modern controllers provide security mechanisms to allow only legitimate firmware to be uploaded, the running code can typically be altered by anyone with network or USB access to the controllers. Thus, despite their responsibility, controllers are vulnerable to several kinds of attacks, including PLC-Blaster worm [7], Ladder Logic Bombs [8], and PLC PIN Control attacks [9].

As a consequence, extra *trusted hardware components* have been proposed to enhance the security of ICS architectures [10], [11]. For instance, McLaughlin [10] proposed to add a policy-based *enforcement mechanism* to mediate the

DOI 10.1109/CSF49147.2020.00025

actuator commands transmitted by the PLC to the physical plant; here, a PLC policy is expressed in terms of some sort of *regular expression*. Mohan et al. [11] introduced a different architecture, in which every PLC runs under the scrutiny of a *monitor* which looks for deviations with respect to *safe behaviours*. If the information obtained via the monitor differs from the expected model(s) of the PLC, a *decision module* is informed to decide whether to pass the control from the "potentially compromised" PLC to a *safety controller* to maintain the plant within the required safety margins.

Both architectures above have been validated by means of simulation-based techniques. However, as far as we know, *formal methodologies* have not been used yet to model and formally verify security-oriented architectures for ICSs.

In this paper, we propose a *formal approach* based on *runtime enforcement* to ensure specification compliance in networks of controllers possibly compromised through *colluding malware* that may forge/drop actuator commands, modify sensor readings, and forge/drop inter-controller communications.

Runtime enforcement [12], [13], [14] is a powerful verification/validation technique, extending *runtime monitoring* [15], [16], [17], and aiming at correcting possibly-incorrect executions of a system-under-scrutiny (SuS). It employs a kind of monitor that acts as a *proxy* between the SuS and the environment interacting with it. At runtime, the monitor *transforms* any incorrect executions exhibited by the SuS into correct ones by either *replacing*, *suppressing* or *inserting* observable actions on behalf of the system. The effectiveness of the enforcement depends on the achievement of the two following general principles [13]:

- *transparency*, *i.e.*, the enforcement must not prevent correct executions of the SuS;
- *soundness*, *i.e.*, incorrect executions of the SuS must be prevented.

Our *goal* is to enforce potentially corrupted controllers using *secure proxies* based on a sub-class of Ligatti's edit automata [13]. These automata will be *synthesised* from enforceable *timed correctness properties* to form networks of *monitored controllers*, as in Figure 2. The proposed enforcement will enjoy both transparency and soundness together with the following features:

- *observation-based monitoring*, *i.e.*, the monitor should only look at the observables of the controller, and not at its internal (possibly obfuscated) executing code;
- *determinism preservation*, *i.e.*, the algorithm to synthesise the monitor should not introduce *nondeterminism*;
- *feasibility*, *i.e.*, the synthesis algorithm should have *polynomial complexity* in the size of the enforced property;
- *deadlock-freedom*, *i.e.*, the enforcement must not introduce deadlocks in the monitored controller;
- *divergence-freedom*, *i.e.*, the enforcement must not introduce divergences in the monitored controller;
- *mitigation* of incorrect/malicious activities, as in Mohan et al.'s safety controller [11];
- *scalability*, *i.e.*, the synthesis algorithm must scale to networks of controllers.

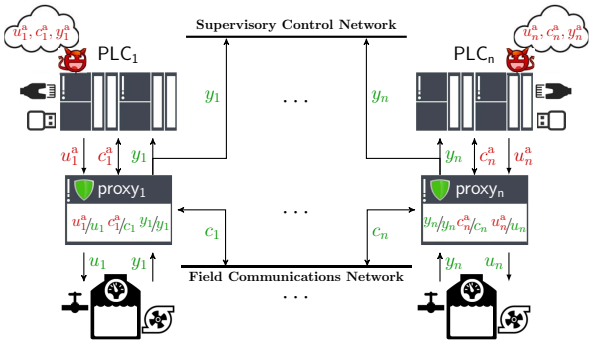

Fig. 2. A network of monitored controllers.

Obviously, when a controller is compromised, these objectives can be achieved only with the introduction of a physically independent *secure proxy*, as advocated by McLaughlin and Mohan et al., which does not have any Internet or USB access, and which is connected with the monitored controller via *secure channels*. This means that the secure proxy should be *bug-free* to avoid possible infiltrations of malware.

This may seem like we just moved the problem over to securing the proxy. However, this is not the case because the proxy only needs to enforce a *timed correctness property* of the system, while the controller does the whole job of controlling the physical process relying on potentially dangerous communications via the Internet or the USB ports. Thus, any upgrade of the control system will be made to the controller and not to the secure proxy. Of course, by no means runtime reconfigurations of the secure proxy should be allowed.

Finally, notice that malicious alterations of sensor signals y_i at network level, or within the sensor devices, is out of the scope of this paper. On the other hand, our architecture depicted in Figure 2 ensures that the sensor measurements transmitted to the *supervisory control network* (*e.g.*, to SCADA devices) will not be corrupted by the controller.

Contribution: Fist of all, we introduce a formal language to specify controller programs. For this very purpose, we resort to *process calculi*, a successful and widespread formal approach in *concurrency theory* for representing complex systems, such as IoT systems [18] and cyber-physical systems [19], and used in many areas, including verification of security protocols [20], [21] and security analysis of *cyber-physical attacks* [22].

We define a simple timed process calculus, based on Hennessy and Regan's *Timed Process Language* (TPL) [23], for specifying controllers, edit automata, and networks of communicating monitored controllers. The proposed edit automata are finite-state and equipped with an ad-hoc semantics for *mitigation*, supporting the capability of emitting correct actions in full autonomy in case the controller is not able to do so.

Then, we define a simple description language to express *timed correctness properties* that should hold upon completion of a finite number of scan cycles of the monitored controller.

This will allow us to abstract over controllers implementations, focusing on general properties which may even be shared by completely different controllers. In this regard, we might resort to one of the several logics existing in the literature for monitoring timed concurrent systems, and in particular cyber-physical systems (see, *i.e.*, Bartocci et al. [24]). Actually, since our formal language is based on Hennessy and Regan's TPL, one might think of using some sort of Hennessy-Milner Logic [25], as in Aceto el al. [26]. However, the peculiar iterative behaviour of controllers convinced us to adopt a simple but expressive sub-class of *regular expressions*, the only properties that under precise conditions can be enforced by finite-state edit automata (see Beauquier et al.'s work [27])[1]. Our regular properties allow us to express interesting correctness properties, spanning over consecutive scan cycles, such as: (i) *timed forward causality*, *e.g.*, if some sensor signals are detected then some actions will occur at some point in the future (communications and/or actuations); (ii) *timed backward causality*, *e.g.*, if some actuations occur then some signals have been previously detected in the past; (iii) *timed mutual exclusion*, *e.g.*, certain events may only occur in mutual exclusion within a certain time interval.

After defining a formal language to describe controller properties, we provide a *synthesis function* $\langle\!|-|\!\rangle$ that, given an alphabet $\mathcal{P}$ of observable actions (sensor readings, actuator commands, and inter-controller communications) and a deterministic regular property e combining events of $\mathcal{P}$, returns, in polynomial time, a syntactically deterministic [28] edit automaton $\langle\!|e|\!\rangle^{\mathcal{P}}$. The resulting enforcement mechanism will ensure the required features mentioned before: observation-based monitoring, transparency, soundness, deadlock-freedom, divergence-freedom, mitigation and scalability.

Notice that, since our enforcement mechanism will be observation-based, *i.e.*, it will observe only observable actions in $\mathcal{P}$, the monitor does not need updates when the enforced controller is reinstalled with an *obfuscated* variant of its code[2] which preserves the observable semantics of the controller.

Last but not least, the same monitor $\langle\!|e|\!\rangle^{\mathcal{P}}$ can be used to enforce different controllers sharing the same observable actions $\mathcal{P}$ and complying with the same enforcing property e.

Outline: Section II contains our formal language to express monitored controllers. Section III provides a non-trivial use case in the context of industrial water treatment systems. Section IV provides a description language for a sub-class of regular properties to express controller behaviours. Section V contains an algorithm to synthesise monitors from regular properties, and the main results of our enforcement. Section VI draws conclusions and discusses related and future work. Technical proofs can be found in the appendix.

II. A Formal language for monitored controllers

In this section, we introduce our *Timed Calculus of Monitored Controllers*, called TCMC, as an extension of Hennessy and Regan's *Timed Process Language* (TPL) [23], to express networks of controllers integrated with edit automata sitting on the network interface of each controller to monitor/correct their interactions with the rest of the system. Like TPL we adopt a *discrete notion of time*: time proceeds in *time slots* separated by tick-actions.

Let us start with some preliminary notation. We use $s, s_k \in \mathsf{Sens}$ to name *sensor signals*; $a, a_k \in \mathsf{Act}$ to indicate *actuator commands*; $c, c_k \in \mathsf{Chn}$ for *channels*; z_1, z_k for *generic names*.

Controller: In our setting, controllers are nondeterministic sequential timed processes evolving through three main phases: *sensing* of sensor signals, *communication* with other controllers, and *actuation*. For convenience, we use five different syntactic categories to distinguish the five main states of a controller: $\mathbb{C}\mathtt{trl}$ for initial states, $\mathbb{S}\mathtt{leep}$ for sleeping states, $\mathbb{S}\mathtt{ens}$ for sensing states, $\mathbb{C}\mathtt{om}$ for communication states, and $\mathbb{A}\mathtt{ct}$ for actuation states. In its initial state, a controller is a recursive process *waiting* for signal stabilisation in order to start the sensing phase:

$$\begin{array}{rcl} \mathbb{C}\mathtt{trl} \ni P & ::= & X \\ \mathbb{S}\mathtt{leep} \ni W & ::= & \mathsf{tick}.W \;\mid\; S \end{array}$$

The main process describing a controller consists of some *recursive process* defined via equations of the form $X = \mathsf{tick}.W$, with $W \in \mathbb{S}\mathtt{leep}$; here, X is a *process variable* that may occur (free) in W. For convenience, our controllers always have at least one initial timed action tick to ensure *time-guarded recursion*, thus avoiding undesired *zeno behaviours*: each recursive call requires at least one time unit. Then, after a determined sleeping period, when sensor signals get stable, the sensing phase can start.

During the sensing phase, the controller waits for a *finite* number of admissible sensor signals. If none of those signals arrives in the current time slot then the controller will *timeout* moving to the following time slot (we adopt the TPL construct $\lfloor\cdot\rfloor\cdot$ for timeout). The syntax is the following:

$$\mathbb{S}\mathtt{ens} \ni S \quad ::= \quad \lfloor \textstyle\sum_{i\in I} s_i.S_i \rfloor S \;\mid\; C$$

where $\sum_{i\in I} s_i.S_i$ denotes the standard construct for nondeterministic choice. Once the sensing phase is concluded, the controller starts its calculations that may depend on *communications* with other controllers governing different physical processes. Controllers communicate with each other for mainly two reasons: either to receive notice about the state of other physical sub-processes or to require an actuation on a physical process which is out of their control. We adopt a *channel-based handshake point-to-point* communication paradigm as in TPL. Notice that, in order to avoid starvation, the communication is always under timeout. The syntax for the communication phase is:

$$\mathbb{C}\mathtt{omm} \ni C \quad ::= \quad \lfloor \textstyle\sum_{i\in I} c_i.C_i \rfloor C \;\mid\; \lfloor \bar{c}.C \rfloor C \;\mid\; A$$

In the actuation phase a controller eventually transmits a *finite* sequence of commands to actuators, and then, it emits a *fictitious control signal* end to denote the end of the scan cycle.

[1] Regular properties have also been used by McLaughlin [10] to express the *security policies* for PLCs in his enforcing monitor C^2.

[2] This is usually what engineers do when a PLC appears to be compromised.

$$\text{(Sleep)}\ \frac{-}{\mathsf{tick}.W \xrightarrow{\mathsf{tick}} W} \qquad \text{(Rec)}\ \frac{X = \mathsf{tick}.W}{X \xrightarrow{\mathsf{tick}} W}$$

$$\text{(ReadS)}\ \frac{j \in I}{\lfloor \sum_{i\in I} s_i.S_i \rfloor S \xrightarrow{s_j} S_j} \qquad \text{(TimeoutS)}\ \frac{-}{\lfloor \sum_{i\in I} s_i.S_i \rfloor S \xrightarrow{\mathsf{tick}} S}$$

$$\text{(InC)}\ \frac{j \in I}{\lfloor \sum_{i\in I} c_i.C_i \rfloor C \xrightarrow{c_j} C_j} \qquad \text{(TimeoutInC)}\ \frac{-}{\lfloor \sum_{i\in I} c_i.C_i \rfloor C \xrightarrow{\mathsf{tick}} C}$$

$$\text{(OutC)}\ \frac{-}{\lfloor \overline{c}.C \rfloor C' \xrightarrow{\overline{c}} C} \qquad \text{(TimeoutOutC)}\ \frac{-}{\lfloor \overline{c}.C \rfloor C' \xrightarrow{\mathsf{tick}} C'}$$

$$\text{(WriteA)}\ \frac{-}{\overline{a}.A \xrightarrow{\overline{a}} A} \qquad \text{(End)}\ \frac{-}{\mathsf{end}.X \xrightarrow{\mathsf{end}} X}$$

TABLE I
LABELLED TRANSITION SYSTEM FOR CONTROLLERS.

After that, the whole scan cycle can restart. Formally,

$$\mathbb{Act} \ni A \quad ::= \quad \overline{a}.A \ \mid\ \mathsf{end}.X$$

Remark 1 (Scan cycle duration and maximum cycle limit): As expected, the signal end must occur well before the *maximum cycle limit* of the controller. We actually work under the assumption that our controllers successfully complete their scan cycle in less than half of the maximum cycle limit. The reasons for this assumption will be clarified in Remark 2.

The operational semantics in Table I is along the lines of Hennessy and Regan's TPL [23].

In the following, we use the metavariable α to range over the set of all observable actions: $\{s, \overline{a}, \overline{c}, c, \mathsf{tick}, \mathsf{end}\}$. These actions denote: sensor readings, actuator commands, channel transmissions, channel receptions, passage of time, and end of scan cycles, respectively.

Monitored controller(s): The core of our runtime enforcement relies on a (timed) sub-class of finite-state Ligatti et al.'s *edit automata* [13], *i.e.*, a particular class of automata specifically designed to modify/suppress/insert actions in a generic system in order to preserve its correct behaviour. The syntax follows:

$$\mathbb{Edit} \ni \mathsf{E} \quad ::= \quad \mathsf{go} \ \mid\ \textstyle\sum_{i\in I} \alpha_i/\beta_i.\mathsf{E}_i \ \mid\ \mathsf{X}$$

The special automaton go will admit any action of the monitored system, while the edit automaton $\sum_{i\in I} \alpha_i/\beta_i.\mathsf{E}_i$ *replaces* actions α_i with β_i, and then continues as E_i, for any $i \in I$, with I finite; here, the metavariables β_i range over the same set of actions seen above for α together with the *non-observable action* τ. Finally, *recursive automata* X are defined via equations of the form $\mathsf{X} = \mathsf{E}$, where the automata variable X may occur (free) in E. The operational semantics of our edit automata is the following:

$$\text{(Go)}\ \frac{-}{\mathsf{go} \xrightarrow{\alpha/\alpha} \mathsf{go}} \qquad \text{(Edit)}\ \frac{j \in I}{\sum_{i\in I} \alpha_i/\beta_i.\mathsf{E}_i \xrightarrow{\alpha_j/\beta_j} \mathsf{E}_j}$$

$$\text{(recE)}\ \frac{\mathsf{X} = \mathsf{E} \quad \mathsf{E} \xrightarrow{\alpha/\beta} \mathsf{E}'}{\mathsf{X} \xrightarrow{\alpha/\beta} \mathsf{E}'}$$

Our *monitored controllers*, written $\mathsf{E} \bowtie \{J\}$, consist of a controller J, for $J \in \mathbb{Ctrl} \cup \mathbb{Sleep} \cup \mathbb{Sens} \cup \mathbb{Comm} \cup \mathbb{Act}$, and an edit automaton E enforcing the behaviour of J, according to the two following transition rules:

$$\text{(Enforce)}\ \frac{J \xrightarrow{\alpha} J' \quad \mathsf{E} \xrightarrow{\alpha/\beta} \mathsf{E}'}{\mathsf{E}\bowtie\{J\} \xrightarrow{\beta} \mathsf{E}'\bowtie\{J'\}}$$

$$\text{(Mitigation)}\ \frac{J \xrightarrow{\mathsf{end}} J' \quad \mathsf{E} \xrightarrow{\alpha/\alpha} \mathsf{E}' \quad \alpha \in \mathsf{Sens} \cup \mathsf{Chn}^* \cup \overline{\mathsf{Act}} \cup \{\mathsf{tick}\}}{\mathsf{E}\bowtie\{J\} \xrightarrow{\alpha} \mathsf{E}'\bowtie\{J\}}$$

The rule (Enforce), inspired by [26], can be used for enforcing *suppressions* (when $\beta = \tau$) or *corrections* (when $\beta \neq \tau$) of *observable actions* α emitted by the controller under scrutiny (we focus on observation-based monitoring).

Thus, in a monitored controller $\mathsf{E} \bowtie \{J\}$ in which the controller J works correctly, the enforcement never occurs (*i.e.*, when applying rule (Enforce) we always have $\alpha = \beta$), and the two components E and J evolve in a tethered fashion, moving through related correct states.

On the other hand, if J is corrupted (for instance, due to the presence of a malware) then E and J may get misaligned within some scan cycle as reaching unrelated states. In this case, the remaining actions emitted by the controller will be suppressed by the monitor until the controller reaches the end of the scan cycle, signalled by the emission of the end-action[3]. Once the compromised controller has been driven to the end of its scan cycle, the transition rule (Mitigation) goes into action.

The rule (Mitigation) allows the *insertion* of a sequence of activities driven by the edit automaton in full autonomy. Intuitively, if the compromised controller signals the end of the scan cycle by emitting the action end and, at the same time, the current edit automaton E is not in the same state, then E will command the execution of a safe trace, without any involvement of the (user program of the) controller, to reach the end of the controller cycle. When both the controller and the edit automaton will be aligned (at the end of the

[3]In general, malware that aims to take control of the plant has no interest in delaying the scan cycle and risking the violation of the maximum cycle limit whose consequence would be the immediate controller shutting down [7].

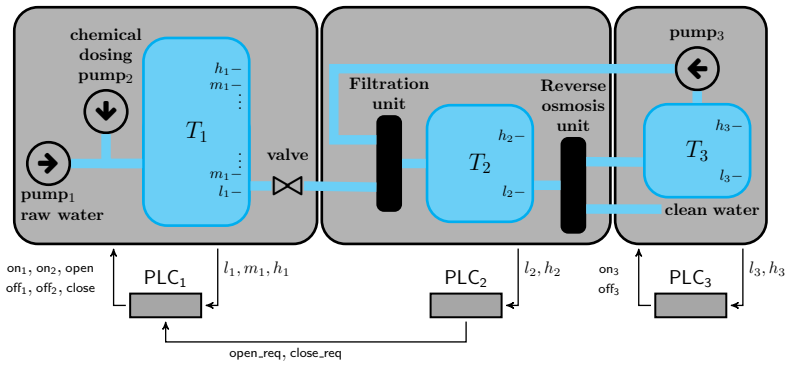

Fig. 3. A simplified Industrial Water Treatment System.

scan cycle) they will synchronise on the action end, via an application of the transition rule (Enforce), and from then on they will continue in a tethered fashion.

Remark 2: The assumption made in Remark 1 ensures us enough time to complete the mitigation of the scan cycle, well before the violation of the maximum cycle limit.

Obviously, we can easily generalise the concept of monitored controller to a *field communications network* of parallel monitored controllers, each one acting on different actuators, and exchanging information via channels. These networks are formally defined via a straightforward grammar:

$$\mathbb{FN}\mathbb{et} \ni N \ ::= \ \mathsf{E}\bowtie\{J\} \ \mid \ N \parallel N$$

with the following operational semantics:

$$\text{(ParL)} \ \frac{N_1 \xrightarrow{\alpha} N_1'}{N_1 \parallel N_2 \xrightarrow{\alpha} N_1' \parallel N_2}$$

$$\text{(ParR)} \ \frac{N_2 \xrightarrow{\alpha} N_2'}{N_1 \parallel N_2 \xrightarrow{\alpha} N_1 \parallel N_2'}$$

$$\text{(ChnSync)} \ \frac{N_1 \xrightarrow{c} N_1' \quad N_2 \xrightarrow{\overline{c}} N_2'}{\begin{array}{l} N_1 \parallel N_2 \xrightarrow{\tau} N_1' \parallel N_2' \\ N_2 \parallel N_1 \xrightarrow{\tau} N_2' \parallel N_1' \end{array}}$$

$$\text{(TimeSync)} \ \frac{N_1 \xrightarrow{\mathsf{tick}} N_1' \quad N_2 \xrightarrow{\mathsf{tick}} N_2' \quad N_1 \parallel N_2 \xrightarrow{\tau} \not\rightarrow}{N_1 \parallel N_2 \xrightarrow{\mathsf{tick}} N_1' \parallel N_2'}$$

Notice that monitored controllers may interact with each other via channel communication (see Rule (ChnSync)). Moreover, via rule (TimeSync) they may evolve in time only when channel synchronisation may not occur (our controllers do not admit zeno behaviours). This ensures *maximal progress* [23], a desirable time property when modelling real-time systems: channel communications will never be postponed.

Having defined the possible actions β of a monitored field network (we recall that β may also range over τ-actions, due to an application of rule (Enforce)), we can easily concatenate actions to define *execution traces*.

Definition 1 (Execution traces): Given a finite execution trace $t = \beta_1 \dots \beta_k$, we write $N \xrightarrow{t} N'$ as an abbreviation for $N = N_0 \xrightarrow{\beta_1} N_1 \xrightarrow{\beta_2} \cdots \xrightarrow{\beta_{k-1}} N_{k-1} \xrightarrow{\beta_k} N_k = N'$.

In the rest of the paper we adopt the following notations.

Notation 1: As usual, we write ϵ to denote the *empty trace*. Given a trace t we write $\mid t \mid$ to denote the *length* of t, *i.e.*, the number of actions occurring in t. Given a trace t we write $\hat{t}$ to denote the trace obtained by removing the τ-actions from t. Given two traces t' and t'', we write $t' \cdot t''$ for the trace resulting from the *concatenation* of t' and t''. For $t = t' \cdot t''$ we say that t' is a *prefix* of t and t'' is a *suffix* of t.

III. Use case: the SWaT System

In this section, we describe how to specify in TCMC a non-trivial network of PLCs to control (a simplified version of) the *Secure Water Treatment system* (SWaT) [29].

SWaT represents a scaled down version of a real-world industrial water treatment plant. The system consists of 6 stages, each of which deals with a different treatment, including: chemical dosing, filtration, dechlorination, and reverse osmosis. For simplicity, in our use case, depicted in Figure 3, we consider only three stages. In the first stage, raw water is *chemically dosed* and pumped in a tank T_1, via two pumps $pump_1$ and $pump_2$. A *valve* connects T_1 with a *filtration unit* that releases the treated water in a second tank T_2. Here, we assume that the flow of the incoming water in T_1 is greater than the outgoing flow passing through the valve. The water in T_2 flows into a *reverse osmosis unit* to reduce inorganic impurities. In the last stage, the water coming from the reverse osmosis unit is either distributed as clean water, if required standards are met, or stored in a backwash tank T_3 and then pumped back, via a pump $pump_3$, to the filtration unit. Here, we assume that tank T_2 is large enough to receive the whole content of tank T_3 at any moment.

The SWaT system has been used to provide a dataset containing physical and network data recorded during 11 days of activity [30]. Part of this dataset contains information about the execution of the system in isolation, while a second part records the effects on the system when exposed to different kinds of cyber-physical attacks. Thus, for instance, (i) *drops* of commands to activate $pump_2$ may affect the quality of the water, as they would affect the correct functioning of the

chemical dosing pump; (ii) *injections* of commands to close the valve between T_1 and T_2, may give rise to an overflow of tank T_1 if this tank is full; (iii) *integrity attacks* on the signals coming from the sensor of the tank T_3 may result in damages of the pump $pump_3$ if it is activated when T_3 is empty.

Each tank has its own PLC, possibly connected with the others via dedicated communication channels. In the rest of the section, we propose a possible implementation of the three PLCs governing the three tanks.

Let us start with the code P_1 of the controller PLC_1 managing the tank T_1. Its definition in TCMC is the following:

$$\begin{aligned} P_1 = \mathsf{tick}.\big(&\lfloor l_1.\lfloor \mathsf{close\_req}.\overline{\mathsf{on_1}}.\overline{\mathsf{on_2}}.\overline{\mathsf{close}}.\mathsf{end}.P_1\rfloor(\overline{\mathsf{on_1}}.\overline{\mathsf{on_2}}.\overline{\mathsf{open}}.\mathsf{end}.P_1)\\ &+ m_1.\lfloor \mathsf{open\_req}.\overline{\mathsf{open}}.\mathsf{end}.P_1 + \mathsf{close\_req}.\overline{\mathsf{close}}.\mathsf{end}.P_1\rfloor(\mathsf{end}.P_1)\\ &+ h_1.\lfloor \mathsf{close\_req}.\overline{\mathsf{off_1}}.\overline{\mathsf{off_2}}.\overline{\mathsf{close}}.\mathsf{end}.P_1\rfloor(\overline{\mathsf{off_1}}.\overline{\mathsf{off_2}}.\overline{\mathsf{open}}.\mathsf{end}.P_1)\rfloor(\mathsf{end}.P_1)\big)\end{aligned}$$

PLC_1 waits for one time slot (to get stable sensor signals) and then checks the water level of the tank T_1, distinguishing between three possible states. If T_1 reaches a low level (signal l_1) then the PLC listens for requests at channel close_req to close the valve between T_1 and T_2, arriving from PLC_2 (the controller of tank T_2). If PLC_1 gets such a request then it turns both pumps on (commands $\overline{\mathsf{on_1}}$ and $\overline{\mathsf{on_2}}$), closes the valve (command $\overline{\mathsf{close}}$), and then returns; otherwise, it times out, turns both pumps on, opens the valve (command $\overline{\mathsf{open}}$), and then returns. If the level of the tank is high (signal h_1) then PLC_1 listens for requests arriving at channel close_req from PLC_2. If a request arrives then the PLC turns both pumps off (commands $\overline{\mathsf{off_1}}$ and $\overline{\mathsf{off_2}}$), it closes the valve, and then returns; otherwise it times out, turns both pumps off, opens the valve, and then returns. Finally, if the tank T_1 is at some intermediate level between l_1 and h_1 (signal m_1) then PLC_1 listens for requests from PLC_2 of opening the valve; if PCL_1 gets an open_req request then it opens the valve, letting the water flow from T_1 to T_2, and returns; otherwise, if it gets a close_req request then it closes the valve, and then returns.

PLC_2 manages the water level of tank T_2. Its code P_2 is defined in TCMC via the following equation:

$$\begin{aligned} P_2 = \mathsf{tick}.(&\lfloor l_2.\lfloor \overline{\mathsf{open\_req}}.\mathsf{end}.P_2\rfloor \mathsf{end}.P_2\\ &+ h_2.\lfloor \overline{\mathsf{close\_req}}.\mathsf{end}.P_2\rfloor \mathsf{end}.P_2\rfloor \mathsf{end}.P_2)\end{aligned}$$

Here, after one time slot, the level of T_2 is checked. If the level is low (signal l_2) then PLC_2 sends a request to PLC_1, via the channel open_req, to open the valve letting the water to flow from T_1 to T_2, and then returns. Otherwise, if the level of tank T_2 is high (signal h_2) then PLC_2 asks PLC_1 to close the valve, via the channel close_req, and then returns.

Finally, PLC_3 manages the water level of tank T_3. Its code P_3 is defined in TCMC via the following equation:

$$P_3 \;=\; \mathsf{tick}.(\lfloor l_3.\overline{\mathsf{off_3}}.\mathsf{end}.P_3 \;+\; h_3.\overline{\mathsf{on_3}}.\mathsf{end}.P_3\rfloor \mathsf{end}.P_3)$$

Here, after one time slot, the level of the backwash tank T_3 is checked. If the level is low (signal l_3) then PLC_3 turns off the pump $pump_3$ (command $\overline{\mathsf{off_3}}$), and then returns. Otherwise, if the level of T_3 is high (signal h_3) then the pump is turned on (command $\overline{\mathsf{on_3}}$) and the whole content of T_3 is pumped back into the filtration unit of T_2; after that the PLC returns.

IV. A FORMAL LANGUAGE FOR CONTROLLER PROPERTIES

In this section, we provide a simple description language to express *correctness properties* that we may wish to enforce at runtime in our controllers. As discussed in the Introduction, we resort to (a sub-class of) *regular properties*, the logical counterpart of regular expressions, as they allow us to express interesting classes of properties referring to one or more scan cycles of a controller. The proposed language distinguishes between two kinds of properties: (i) *global properties*, $e \in \mathbb{Prop}\mathbb{G}$, to express general controllers' execution traces; (ii) *local properties*, $p \in \mathbb{Prop}\mathbb{L}$, to express traces confined to a finite number of consecutive scan cycles. The two families of properties are formalised via the following regular grammar:

$$\begin{array}{lll} e \in \mathbb{Prop}\mathbb{G} & ::= & p^* \\ p \in \mathbb{Prop}\mathbb{L} & ::= & \epsilon \mid p_1; p_2 \mid \cup_{i\in I}\pi_i.p_i \end{array}$$

where $\pi_i \in \mathsf{Sens} \cup \overline{\mathsf{Act}} \cup \mathsf{Chn}^* \cup \{\mathsf{tick}\} \cup \{\mathsf{end}\}$ denote *atomic properties*, sometimes called *events*, that may occur as prefix of a property. With an abuse of notation, we use the symbol ϵ to denote both the *empty property* and the *empty trace*.

The *semantics* of our logic is naturally defined in terms of sets of execution traces which satisfy a given property; its formal definition is given in Table II.

However, the syntax of our logic is a bit too permissive with respect to our intentions, as it allows us to describe partial scan cycles, *i.e.*, cycles that have not completed. Thus, we restrict ourselves to considering properties which builds on top of local properties associated to *complete scan cycles*, *i.e.*, scan cycles whose last action is an end-action. Formally,

Definition 2: Well-formed properties are defined as follows:

- the local property $\mathsf{end}.\epsilon$ is well formed;
- a local property of the form $p_1; p_2$ is well formed if p_2 is well formed;
- a local property of the form $\cup_{i\in I}\pi_i.p_i$ is well formed if, for any $j \in I$ either $\pi_j.p_j = \mathsf{end}.\epsilon$ or, p_j is well formed.

A global property p^* is well-formed if p is well-formed.

In the rest of the paper, we adopt the following notations.

Notation 2: We omit trailing empty properties, writing π instead of $\pi.\epsilon$. For $k > 0$, we write $\pi^k.p$ as a shorthand for $\pi.\pi...\pi.p$, where prefix π appears k consecutive times. Given a local property p we write $\mathsf{events}(p)$ to denote the set of events occurring in p; for a global property $e = p^*$, $\mathsf{events}(e)$ is given by $\mathsf{events}(p)$. Given a set of events $\mathcal{A}$ and a local property p, we use $\mathcal{A}$ itself as an abbreviation for the property $\cup_{\pi\in\mathcal{A}}\pi.\epsilon$, and $\mathcal{A}.p$ as an abbreviation for the property $\cup_{\pi\in\mathcal{A}}\pi.p$. Given a set of events $\mathcal{A}$, with $\mathsf{end} \notin \mathcal{A}$, we write $\mathcal{A}^{\leq k}$, for $k \geq 0$, to denote the property defined as follows:

- $\mathcal{A}^{\leq 0} \triangleq \mathsf{end}$
- $\mathcal{A}^{\leq k} \triangleq \mathsf{end} \cup \mathcal{A}.\mathcal{A}^{\leq k-1}$, for $k > 0$.

Thus, the property $\mathcal{A}^{\leq k}$ captures all possible sequences of events of $\mathcal{A}$ whose length is at most k, for $k \in \mathbb{N}$.

A. Some significant correctness properties

In this section, we describe three different classes of correctness properties, expressible in our language, which are suitable

$$
\begin{array}{lcl}
[\![p^*]\!] & \triangleq & \{\epsilon\} \cup \bigcup_{n \in \mathbb{N}^+} \{t \mid t = t_1 \cdot \ldots \cdot t_n, \text{ with } t_i \in [\![p]\!], \text{ for } 1 \leq i \leq n\} \\
[\![\epsilon]\!] & \triangleq & \{\epsilon\} \\
[\![p_1; p_2]\!] & \triangleq & \{t \mid t = t_1 \cdot t_2, \text{ with } t_1 \in [\![p_1]\!] \text{ and } t_2 \in [\![p_2]\!]\} \\
[\![\bigcup_{i \in I} \pi_i.p_i]\!] & \triangleq & \bigcup_{i \in I} \{t \mid t = \pi_i \cdot t', \text{ with } t' \in [\![p_i]\!]\}
\end{array}
$$

TABLE II
TRACE SEMANTICS OF OUR REGULAR PROPERTIES.

to describe significant controller properties spanning over a finite number of scan-cycles: *timed forward causality*, *timed backward causality* and *timed mutual exclusion*.

In order to properly define these three classes of properties, we introduce four predicates relating events, properties and time (*i.e.*, scan cycles). Thus, given an event π, a local property p and a natural number n, we write π $pfx_{\mathrm{n}}^{\forall}$ p (*resp.*, π $sfx_{\mathrm{n}}^{\forall}$ p) to say that for any trace t in $[\![p]\!]$ the action π (associated to the event π) must appear in some *prefix* (*resp.*, *suffix*) of t, within at most n scan cycles. As the end of a scan cycle is always represented via the action end, this means that in all traces in $[\![p]\!]$ the action π is always preceded (*resp.*, followed) by at most $n-1$ end-actions. Formally,

Definition 3: Let $p \in \mathbb{Pr}\mathbb{o}\mathbb{p}\mathbb{L}$ be a well-formed local property, π be an event, and $n \in \mathbb{N}$ be a natural number. The predicate π $pfx_{\mathrm{n}}^{\forall}$ p (*resp.*, π $sfx_{\mathrm{n}}^{\forall}$ p) returns true only if for any $t \in [\![p]\!]$ there exists a trace $t' \cdot \mathsf{end}$, which is a prefix (*resp.*, suffix) of t, such that the action π occurs in t' and the action end occurs in t' at most $n-1$ times.

Similarly, given an event π, a local property p and a natural number n we write π $pfx_{\mathrm{n}}^{\exists}$ p (*resp.*, π $sfx_{\mathrm{n}}^{\exists}$ p) to prescribe that the action π (associated to the event π) must appear in some *prefix* (*resp.*, *suffix*) of at least one trace in $[\![p]\!]$, within at most n scan cycles. Formally,

Definition 4: Let $p \in \mathbb{Pr}\mathbb{o}\mathbb{p}\mathbb{L}$ be a well-formed local property, π be an event, and $n \in \mathbb{N}$ be a natural number. The predicate π $pfx_{\mathrm{n}}^{\exists}$ p (*resp.*, π $sfx_{\mathrm{n}}^{\exists}$ p) returns true only if there is a trace $t \in [\![p]\!]$ and a trace $t' \cdot \mathsf{end}$, which is a prefix (*resp.*, suffix) of t, such that the action π occurs in t' and the action end occurs in t' at most $n-1$ times.

Now, everything is in place to define our classes of properties. In what follows, the variables π_1, π_2 and π_i, for $i \in I$, will range over *untimed events*, *i.e.*, $\pi_1, \pi_2, \pi_i \in \mathsf{Sens} \cup \overline{\mathsf{Act}} \cup \mathsf{Chn}^*$.

Timed forward causality: In this class we collect properties expressing the causality between a triggering event π_1 and a second event π_2 that will occur at some point in the future. For instance, if an event π_1 occurs (*e.g.*, a sensor signal) then a subsequent event π_2 (*e.g.*, some communication or actuation) will occur in the future, for the first time, within a specific time interval, *e.g.*, *after m scan cycles and before n scan cycles*, with $0 \leq m \leq n \in \mathbb{N}$. For simplicity, we do not allow nesting between causes and effects, *i.e.*, once a triggering event π_1 has occurred it may occur again only after an occurrence of the event π_2.[4] Thus, we write π_1 $entails_{[m,n]}$ π_2 to denote a class of properties of the form $(p; (q \cup \pi_1.r))^* \in \mathbb{Pr}\mathbb{o}\mathbb{p}\mathbb{G}$, for $p, q, r \in \mathbb{Pr}\mathbb{o}\mathbb{p}\mathbb{L}$ such that:

- the event π_1 does not occur neither in p nor in q (this because if the event π_1 does not occur at all then the whole property is trivially satisfied);
- the event π_1 does not occur in r (we do not allow nested causality);
- for $m > 0$ it holds that $\neg(\pi_2$ $pfx_{\mathrm{m-1}}^{\exists}$ $r)$, *i.e.*, if the event π_1 occurs then π_2 never occurs in the following $m-1$ scan cycles captured by r (including the current one);
- π_2 $pfx_{\mathrm{n}}^{\forall}$ r, *i.e.*, if the event π_1 occurs then the event π_2 must always occur within at most n scan cycles captured by r (including the current scan cycle).

As an example, we might enforce a timed forward property in PLC_1 of our use case to prevent water overflow in the tank T_2, due to a misuse of the valve connecting the tanks T_1 and T_2. Assuming that $z \in \mathbb{N}$ is the time (expressed in scan cycles) required to overflow the tank T_2, we might consider to enforce a timed forward property of the form:

$$\mathsf{open\_req}\ entails_{[0,w]}\ \overline{\mathsf{close}}$$

with $w << z$, saying that if the PLC receives a request to open the valve (*i.e.*, the event $\mathsf{open\_req}$ occurs) then the valve will be eventually closed (the event $\overline{\mathsf{close}}$ will eventually occur) within at most w scan cycles (including the current one). This ensures us that the valve will remain open at most w scan cycles, with $w << z$, preventing the overflow of T_2. As the scan cycle of PLC_1 is at most 6 actions long, a possible implementation of this property is the following:

$$\left(\mathsf{tick}.\mathcal{A}; (\mathcal{A}^{\leq 5} \cup \mathsf{open\_req}.\mathcal{B}_4^z)\right)^*,$$

with $\mathcal{A} = \{l_1, m_1, h_1\} \cup \{\overline{\mathsf{on_1}}, \overline{\mathsf{on_2}}, \overline{\mathsf{off_1}}, \overline{\mathsf{off_2}}, \overline{\mathsf{open}}, \overline{\mathsf{close}}\} \cup \{\mathsf{close\_req}\} \cup \{\mathsf{tick}\}$ being the set of all possible actions of PLC_1 except for $\mathsf{open\_req}$ and end, while $\mathcal{B}_h^k$ is the set of all admitted actions in the following k scan cycles, where h counts intra-scan-cycles actions. Formally,

- $\mathcal{B}_h^k \triangleq \overline{\mathsf{close}}.\mathcal{A}^{\leq h-1} \cup (\mathcal{A} \backslash \{\overline{\mathsf{close}}\}).\mathcal{B}_{h-1}^k \cup\ \mathsf{end.tick}.\mathcal{B}_6^{k-1}$, for $k > 1$, and $h > 0$;
- $\mathcal{B}_0^k \triangleq \mathsf{end.tick}.\mathcal{B}_6^{k-1}$, for $k > 1$; $\mathcal{B}_0^1 \triangleq \overline{\mathsf{close}}.\mathsf{end}$.

Timed backward causality: In this class we collect properties that express the causality between a triggering event π_1 and a second event π_2 that occurred back in the past. For instance, if an event π_1 occurs (*e.g.*, an actuator command) then an event π_2 (*e.g.*, either some sensor signal or some communication) must have occurred in the past *at least m scan cycles earlier but no more than n scan cycles earlier*. Again, for simplicity we do not consider nesting between causes π_1

[4]This is a reasonable requirement in cyber-physical systems, where physical processes are sensed only after some proper actuation has been completed.

$$
\begin{array}{rcl}
\langle\!|\, p^* \,|\!\rangle^{\mathcal{P}} & \triangleq & \mathsf{X},\ \text{for } \mathsf{X} = \langle\!|\, p \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}} \\
\langle\!|\, \epsilon \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \mathsf{X} \\
\langle\!|\, p_1; p_2 \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \langle\!|\, p_1 \,|\!\rangle^{\mathcal{P}}_{\mathsf{Z}},\ \text{for } \mathsf{Z} = \langle\!|\, p_2 \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}},\ \mathsf{Z} \text{ fresh} \\
\langle\!|\, \bigcup_{i\in I} \pi_i.p_i \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \mathsf{Z},\ \text{for } \mathsf{Z} = \sum_{i\in I} \pi_i/\pi_i.\langle\!|\, p_i \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}} + \sum_{\alpha\in\mathcal{P}\setminus(\cup_{i\in I}\pi_i\cup\{\mathsf{tick},\mathsf{end}\})} \alpha/\tau.\mathsf{Z},\ \mathsf{Z} \text{ fresh}
\end{array}
$$

TABLE III
MONITOR SYNTHESIS FROM OUR REGULAR PROPERTIES.

and effects π_2. Thus, we write $\pi_1\ requires_{[m,n]}\ \pi_2$ to denote a class of properties of the form $(s; (p \cup (q; \pi_1.r)))^* \in \mathbb{P}\mathrm{r}\circledcirc\mathbb{p}\mathbb{G}$, for $p, q, r, s \in \mathbb{P}\mathrm{r}\circledcirc\mathbb{p}\mathbb{L}$, such that:

- π_1 does not occur neither in s nor in p (if π_1 does not occur at all then the property is trivially satisfied);
- for $m > 0$ it holds that $\neg(\pi_2\ sfx^{\exists}_{\mathrm{m}-1}\ q)$, *i.e.*, if the event π_1 occurs then the event π_2 never occurred in the previous $m-1$ scan cycles captured by q;
- $\pi_2\ sfx^{\forall}_{\mathrm{n}}\ q$, *i.e.*, if the event π_1 occurs then π_2 must have occurred in the past, at most n scan cycles earlier.

As an example, we might enforce a timed backward property in PLC_3 of our use case to prevent damages in the pump $pump_3$ due to lack of water in the tank T_3. Thus, we might consider to enforce a property of the form:

$$\overline{\mathsf{on_3}}\ requires_{[0,0]}\ h_3$$

saying that if the $pump_3$ is on then the level of the tank T_3 must have been (previously) sensed as high in the current scan cycle, preventing pump damages. As the scan cycle of PLC_3 is at most 2 actions long, a possible implementation of this property is:

$$(\mathsf{tick}; (l_3.\mathcal{A}^{\leq 1} \cup h_3; \overline{\mathsf{on_3}}.\mathsf{end}))^*$$

where $\mathcal{A} = \{l_3, h_3\} \cup \{\overline{\mathsf{off_3}}\} \cup \{\mathsf{tick}\}$ is the set of all possible actions of PLC_3 except for $\mathsf{on_3}$ and end.

Timed mutual exclusion: In this class we collect properties denoting that certain events π_i, for $i \in I$, may occur only in mutual exclusion within n consecutive scan cycles.

Thus, we write $mutual\_exclusion_{[n]}\mathcal{A}$, with $n \geq 0$ and $\mathcal{A} = \cup_{i\in I}\pi_i$ to denote a class of properties of the form: $\big(p; (q \cup (\cup_{i\in I}\pi_i.r_i))\big)^* \in \mathbb{P}\mathrm{r}\circledcirc\mathbb{p}\mathbb{G}$, for $p, q, r_i \in \mathbb{P}\mathrm{r}\circledcirc\mathbb{p}\mathbb{L}$, where:

- for all $i \in I$ the event π_i does not occur neither in p nor in q (if π_i does not occur at all then the property is trivially satisfied);
- for all $i \in I$ the event π_i does not occur in r_i (we do not consider nesting of properties);
- for all $i \in I$, all traces in $[\![r_i]\!]$ are n scan cycles long, *i.e.*, they contain n occurrences of the end action;
- $\neg(\pi_i\ pfx^{\exists}_{\mathrm{n}}\ \pi_j.r_j)$, for all $i, j \in I$, with $i \neq j$ (the events π_i are in mutual exclusion for n consecutive scan cycles).

As an example, we might enforce a timed mutual exclusion property in the PLC_2 of our use case to prevent chattering of the valve, *i.e.*, rapid opening and closing which may cause mechanical failures on the long run. Thus, we might consider to enforce a property of the form:

$$mutual\_exclusion_{[3]}\{\overline{\mathsf{open\_req}}, \overline{\mathsf{close\_req}}\}$$

saying that the events to request the opening and the closing of the valve (events $\overline{\mathsf{open\_req}}$ and $\overline{\mathsf{close\_req}}$, respectively) may only occur in mutual exclusion within 3 consecutive scan cycles. As the scan cycle of PLC_2 has at most 2 actions, a possible implementation of the property is the following:

$$(\mathsf{tick}.\mathcal{A}; (\overline{\mathsf{open\_req}}.\mathsf{end}.\mathcal{B}^2 \cup \overline{\mathsf{close\_req}}.\mathsf{end}.\mathcal{B}^2))^*$$

where $\mathcal{A}{=}\{l_2, h_2\} \cup \{\mathsf{tick}\}$ is the set of all possible actions of PLC_2 except for $\overline{\mathsf{open\_req}}$, $\overline{\mathsf{close\_req}}$, and end, while $\mathcal{B}^k$ is the set of all admitted actions in the following k scan cycles. Formally, $\mathcal{B}^k \triangleq \mathsf{tick}.\mathcal{A}^{\leq 2}; \mathcal{B}^{k-1}$, for $k > 0$, and $\mathcal{B}^0 \triangleq \epsilon$.

V. MONITOR SYNTHESIS

In this section, we provide an algorithm to synthesise monitors from regular properties of the kind defined in the previous section. In particular, given (i) a set $\mathcal{P}$ of observable actions (*i.e.*, different from τ-actions), and (ii) a global property $e \in \mathbb{P}\mathrm{r}\circledcirc\mathbb{p}\mathbb{G}$ whose events are contained in (the set of events associated to) $\mathcal{P}$, the synthesis returns an edit automaton that is capable to enforce (the preservation of) the property e during the execution of a generic controller whose actions are contained in $\mathcal{P}$. As we distinguish global properties from local ones, we define our synthesis algorithm in two steps.

The synthesis algorithm is defined in Table III by induction on the structure of the property given in input. The monitor $\langle\!|\, p^* \,|\!\rangle^{\mathcal{P}}$ associated to a global property p^* and a set of actions $\mathcal{P}$, is an edit automaton defined via the recursive equation $\mathsf{X} = \langle\!|\, p \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, to enforce the local property p during each scan cycle via the edit automaton resulting from the synthesis $\langle\!|\, p \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, parametric in the automata variable X and the set of actions $\mathcal{P}$.

The edit automaton $\langle\!|\, p_1; p_2 \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}}$ associated to the property $p_1; p_2$ is given by the sequential composition of the corresponding edit automata. The two automata are composed by replacing in $\langle\!|\, p_1 \,|\!\rangle^{\mathcal{P}}_{\mathsf{Z}}$ the free edit variable Z, where $\mathsf{Z} \neq \mathsf{X}$, with the continuation $\langle\!|\, p_2 \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}}$. Finally, the edit automaton associated to a union property $\cup_{i\in I}\pi_i.p_i$ permits all actions associated to the events π_i, and suppresses all the others.

Remark 3 (Synthesis vs. mitigation): Notice that our synthesised enforcers never suppress tick-actions and end-actions. In particular, end-actions are crucial watchdogs signalling the end of a controller scan cycle: if the enforcer is in line with the controller then a new scan cycle is free to start, otherwise, if this is not the case, the enforcer launches a mitigation cycle by yielding some correct trace, without any involvement of the controller, to reach the completion of the current scan cycle.

Our synthesis algorithm allows us to define an enforcement mechanism that ensures the features stated in the Introduc-

$$
\begin{array}{rcl}
\langle\!|\ \mathcal{D}^{\leq k}\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \mathsf{D}^k,\ \text{for } \mathsf{D}^k = \sum_{\alpha\in\mathcal{D}} \alpha/\alpha.\langle\!|\ \mathcal{D}^{\leq k-1}\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}} + \sum_{\alpha\in\mathcal{P}\setminus(\mathcal{D}\cup\{\mathsf{tick},\mathsf{end}\})} \alpha/\tau.\mathsf{D}^k \\
\langle\!|\ \mathcal{D}^{\leq 0}\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \mathsf{D}^0,\ \text{for } \mathsf{D}^0 = {}^{\mathsf{end}}/_{\mathsf{end}}.\langle\!|\ \epsilon\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}} + \sum_{\alpha\in\mathcal{P}\setminus\{\mathsf{tick},\mathsf{end}\}} \alpha/\tau.\mathsf{D}^0 \\
\langle\!|\ \epsilon\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \mathsf{X}
\end{array}
$$

TABLE IV
SYNTHESIS OF THE GENERIC PROPERTY $\mathcal{D}^{\leq k}$, FOR $k \geq 0$.

tion: *observation-based monitoring*, *determinism preservation*, *feasibility*, *transparency*, *soundness*, *deadlock-freedom*, *divergence-freedom*, and *scalability*.

Let us formally prove these requirements. In the following, with a small abuse of notation, given a set of actions $\mathcal{P}$, we will use $\mathcal{P}$ to denote also the set of the corresponding events.

Our enforcing monitoring is trivially observation-based as our edit automata admit only correcting actions of the form α/β, in which the metavariable α, by definition, cannot be the non-observable action τ.

As concerns determinism preservation, we focus on Aceto et al.'s syntactic notion of deterministic edit automata [28].

Definition 5: An edit automaton E is called *syntactically deterministic* if for any sub-term $\sum_{i\in I} {}^{\alpha_i}/_{\beta_i}.\mathsf{E}_i$ occurring in E it holds that $\alpha_k \neq \alpha_h$, for $k, h \in I$ and $k \neq h$.

Now, by inspection on the definition of $\langle\!|\ \cup_{i\in I}\pi_i.p_i\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$ our synthesis algorithm does not introduce nondeterminism. The proof of the following result is by induction on the structure of the enforced property.

Proposition 1 (Determinism preservation): Given a global property $e \in \mathbb{P}\mathrm{rop}\mathbb{G}$ and a set of actions $\mathcal{P}$, the edit automaton $\langle\!|e|\!\rangle^{\mathcal{P}}$ is syntactically deterministic.

The complexity of the algorithm is polynomial on the size of the set $\mathcal{P}$ and the dimension of the enforced property e; where, intuitively, the dimension of e, written $\mathsf{dim}(e)$, is given by the number of operators occurring in it.

Proposition 2 (Polynomial Complexity): Given a property $e \in \mathbb{P}\mathrm{rop}\mathbb{G}$ and a set of actions $\mathcal{P}$ such that $\mathsf{events}(e) \subseteq \mathcal{P}$, the complexity of the algorithm to synthesise $\langle\!|e|\!\rangle^{\mathcal{P}}$ is $\mathcal{O}(m{\cdot}n)$, with $m = \mathsf{dim}(e)$ and n being the size of the set $\mathcal{P}$.

Let us move to the next required property: *transparency*. Intuitively, the enforcement induced by a property $e \in \mathbb{P}\mathrm{rop}\mathbb{G}$ should not prevent those traces of the controller under scrutiny satisfying the property e itself [13].

Theorem 1 (Transparency): Let $e \in \mathbb{P}\mathrm{rop}\mathbb{G}$ and $\mathcal{P}$ be a set of observable actions such that $\mathsf{events}(e) \subseteq \mathcal{P}$. Let $P \in \mathbb{C}\mathrm{tr}\mathbb{l}$ be a controller whose actions are contained in $\mathcal{P}$. Let t be a trace of $\mathsf{go}{\bowtie}\{P\}$. If $t \in [\![e]\!]$ then t is a trace of $\langle\!|e|\!\rangle^{\mathcal{P}} \bowtie \{P\}$.

Another important property of our enforcement is *soundness*. Intuitively, a monitored controller yields only execution traces which satisfy the enforced property.

Theorem 2 (Soundness): Let $e \in \mathbb{P}\mathrm{rop}\mathbb{G}$ be a well-formed global property and $\mathcal{P}$ be a set of observable actions such that $\mathsf{events}(e) \subseteq \mathcal{P}$. Let $P \in \mathbb{C}\mathrm{tr}\mathbb{l}$ be a controller whose actions are contained in $\mathcal{P}$. If t is a trace of the system $\langle\!|e|\!\rangle^{\mathcal{P}} \bowtie \{P\}$ then $\hat{t}$ is a prefix of some trace in $[\![e]\!]$ (for $\hat{t}$, see Notation 1).

Here, it is important to stress that in general soundness does not ensure deadlock-freedom of the monitored controller. That is, it may be possible that the enforcement of some property e causes a deadlock of the controller P under scrutiny. In particular, this may happen in our controllers only when the initial sleeping phase does not match the enforcing property. Intuitively, a local property will be called a k-sleeping property only if it allows k initial time instants of sleep.

Definition 6: For $k \in \mathbb{N}^+$, we say that $p \in \mathbb{P}\mathrm{rop}\mathbb{L}$ is a *k-sleeping local property*, only if $[\![p]\!] = \{t \mid t = t_1 \cdot \ldots \cdot t_n,\ \text{for } n > 0,\ \text{s.t. } t_i = \mathsf{tick}^k{\cdot}t_i'{\cdot}\mathsf{end},\ \mathsf{end} \notin t_i',\ \text{and } 1 \leq i \leq n\}$. We say that p^* is a *k-sleeping global property* only if p is.[5]

The enforcement of k-sleeping properties does not introduce deadlocks in k-sleeping controllers.

Proposition 3 (Deadlock-freedom): Let $e = p^*$, for $p \in \mathbb{P}\mathrm{rop}\mathbb{L}$, be a k-sleeping property, and $\mathcal{P}$ be a set of observable actions such that $\mathsf{events}(e) \subseteq \mathcal{P}$. Let $P \in \mathbb{C}\mathrm{tr}\mathbb{l}$ be a controller of the form $P = \mathsf{tick}^k.S$ whose set of observable actions is contained in $\mathcal{P}$. Then, $\langle\!|e|\!\rangle^{\mathcal{P}} \bowtie \{P\}$ does not deadlock.

Another important property of our enforcement mechanism is *divergence-freedom*. In practise, the enforcement does not introduce divergence: monitored controllers will always be able to complete their scan cycles by executing a finite number of actions. This is because in our enforcing regular properties e the number of events within two subsequent end events is always finite.[6]

Proposition 4 (Divergence-freedom): Let $e \in \mathbb{P}\mathrm{rop}\mathbb{G}$ be a well-formed global property and $\mathcal{P}$ be a set of observable actions. Let $P \in \mathbb{C}\mathrm{tr}\mathbb{l}$ be a controller whose set of observable actions is contained in $\mathcal{P}$. Then, there exists a $k \in \mathbb{N}^+$ such that whenever $\langle\!|e|\!\rangle^{\mathcal{P}} \bowtie \{P\} \xrightarrow{t} \mathsf{E} \bowtie \{J\}$, if $\mathsf{E} \bowtie \{J\} \xrightarrow{t'} \mathsf{E}' \bowtie \{J'\}$, with $\mid t' \mid \geq k$, then $\mathsf{end} \in t'$.

Finally, the soundness of our runtime enforcement scale to *field communications networks* of controllers. Intuitively, the soundness of a monitored controller is preserved when running in parallel with other controllers in the same field communications network. By an application of Theorem 2 we have:

Corollary 1 (Scalability): Let $e \in \mathbb{P}\mathrm{rop}\mathbb{G}$ be a well-formed global property and $\mathcal{P}$ be a set of observable actions such that $\mathsf{events}(e) \subseteq \mathcal{P}$. Let $N \in \mathbb{FN}\mathrm{et}$ be a field network and $P \in \mathbb{C}\mathrm{tr}\mathbb{l}$ be a controller whose set of observable actions is contained in $\mathcal{P}$. If $(\langle\!|e|\!\rangle^{\mathcal{P}} \bowtie \{P\}) \parallel N \xrightarrow{t} (\mathsf{E} \bowtie \{J\}) \parallel N'$, for some t, E, J and N', then whenever $\langle\!|e|\!\rangle^{\mathcal{P}} \bowtie \{P\} \xrightarrow{t'} \mathsf{E} \bowtie \{J\}$ the trace $\widehat{t'}$ is a prefix of some trace in $[\![e]\!]$.

As an example, we show an application of Corollary 1 to the field network consisting of the three PLCs of our case

[5]It is easy to see that k-sleeping properties are always well-formed.
[6]Technically speaking, the edit automaton $\langle\!|e|\!\rangle^{\mathcal{P}}$ may not diverge.

study, enforced by the three properties given in Section IV , respectively:

- $e_1 \triangleq \big(\mathsf{tick}.\mathcal{A}; (\mathcal{A}^{\leq 5} \cup \mathsf{open\_req}.\mathcal{B}_4^z)\big)^*$, the timed forward causality property for PLC_1, whose corresponding edit automaton is synthesised in Table V;
- $e_2 \triangleq (\mathsf{tick}.\mathcal{A}; (\overline{\mathsf{open\_req}}.\mathsf{end}.\mathcal{B}^2 \cup \overline{\mathsf{close\_req}}.\mathsf{end}.\mathcal{B}^2))^*$ the timed mutual exclusion property for PLC_2, whose corresponding edit automaton is synthesised in Table VI;
- $e_3 \triangleq (\mathsf{tick}; (l_3.\mathcal{A}^{\leq 1} \cup h_3; \overline{\mathsf{on_3}}.\mathsf{end}))^*$ the timed backward causality property for the controller PLC_3, whose corresponding edit automaton is synthesised in Table VII.

The three syntheses above rely on the synthesis of general properties of the form $\mathcal{D}^{\leq k}$, for an arbitrary set of events $\mathcal{D}$, given in Table IV.

A straightforward application of Corollary 1 follows:

Proposition 5: Let $\prod_{i=1}^{3} \langle\!|e_i|\!\rangle \bowtie \{\mathrm{PLC}_i\} \xrightarrow{t} \prod_{i=1}^{3} \mathsf{E}_i \bowtie \{J_i\}$ be an arbitrary trace of the whole monitored network. Then, for any $1 \leq i \leq 3$, if $\langle\!|e_i|\!\rangle \bowtie \{\mathrm{PLC}_i\} \xrightarrow{t_i} \mathsf{E}_i \bowtie \{J_i\}$ then $\widehat{t_i}$ is a prefix of some trace in $[\![e_i]\!]$.

VI. Conclusions, related and future work

We have defined a formal language to express networks of monitored controllers, potentially compromised with colluding malware that may forge/drop actuator commands, modify sensor readings, and forge/drop inter-controller communications.

The runtime enforcement has been achieved via a finite-state sub-class of Ligatti's edit automata equipped with an ad-hoc operational semantics to deal with *system mitigation*, by inserting actions in full autonomy when the monitored controller is not able to do so in a correct manner.

Then, we have defined a simple description language to express ad-hoc *timed regular properties* which have been used to describe both causality and mutual exclusion of events laying in intervals of time expressed in terms of scan cycles of the monitored controller. Some sort of regular properties have already been used by McLaughlin [10] to express *security policies* for PLCs in his enforcing monitor C^2.

Once defined a formal language to describe controller properties, we have provided a *synthesis function* $\langle\!|-|\!\rangle$ that, given an alphabet $\mathcal{P}$ of observable actions (sensor readings, actuator commands, and inter-controller communications) and a deterministic regular property e, where the events in e are part of the alphabet $\mathcal{P}$, returns, in a time which is *polynomial* in the sizes of $\mathcal{P}$ and e, a *syntactically deterministic* and *observation-based* edit automaton $\langle\!|e|\!\rangle^{\mathcal{P}}$; here observation-based means that the monitoring acts only on those observable actions occurring in $\mathcal{P}$. The resulting enforcement mechanism will ensure the required features advocated in the Introduction: transparency, soundness, deadlock-freedom, divergence-freedom, mitigation and scalability. In particular, with regards to mitigation, as reported in Remark 3, our synthesised enforcers never suppress end-actions as they are crucial watchdogs signalling the end of a controller scan cycle: if the enforcer is aligned with the controller then a new scan cycle is free to start, otherwise, if this is not the case, the enforcer launches a mitigation cycle by yielding some correct trace, without any involvement of the controller, to reach the completion of the current scan cycle.

Notice that the same monitor $\langle\!|e|\!\rangle^{\mathcal{P}}$ can be used to enforce different controllers sharing the same observable actions $\mathcal{P}$ and complying with the same enforcing property e (both the controller and the property e must agree on the duration of the initial sleeping phase). Furthermore, in the Introduction we have carefully argued about the advantages of securing the monitoring proxy rather than the controller itself.

Finally, an exemplification of our enforcement mechanism has been provided by means of a non-trivial running example in the context of industrial water treatment systems.

Related work: The notion of *runtime enforcement* was introduced by Schneider [12] to enforce security policies. These properties are enforced by means of *security automata*, a kind of automata that terminates the monitored system in case of violation of the property.

Ligatti et al. [13] extended Schneider's work by proposing the notion of *edit automata*, *i.e.*, an enforcement mechanism able of replacing, suppressing, or even inserting system actions. In general, Ligatti et al.'s edit automata have an enumerable number of states, whereas in the current paper we restrict ourselves to finite-state edit automata. Furthermore, in its original definition the insertion of actions is possible at any moment, whereas our monitoring edit automata can insert actions, via the rule (Mitigation), only when the controller under scrutiny reaches a specific state, *i.e.*, the end of the scan cycle. We also use correcting actions of the form α/β, in the style of Aceto et al. [26]. These actions can be easily expressed in the original Ligatti's formulation by inserting the action β first, and then suppressing the action α.

Bielova [31] provided a stronger notion of enforceability equipped with a *predictability* criterion to prevent monitors from transforming invalid executions in an arbitrary manner. Intuitively, a monitor is said predictable if one can predict the number of transformations used to correct invalid executions, thereby avoiding unnecessary transformations.

Falcone et al. [32], [14] proposed a synthesis algorithm, relying on Street automata, to translate most of the property classes defined within the *Safety-Progress hierarchy* [33] into enforcers. In the Safety-Progress classification our global properties can be seen as *guarantee properties* for which all execution traces that satisfy a property contain at least one prefix that still satisfies the property.

Beauquier et al. [27] proved that finite-state edit automata (*i.e.* those edit automata we are actually interested in) can only enforce a sub-class of regular properties. Actually they can enforce *all and only* the regular properties that can be recognised by a finite automata whose cycles always contain at least one final state. This is the case of our enforced regular properties, as well-formed local properties in $\mathbb{P}\mathrm{rop}\mathbb{L}$ always terminate with the "final" atomic property end.

Some interesting results on runtime enforcement of *reactive systems* (which have many aspects in common with control systems) have been presented by Könighofer et al. [34]. They defined a synthesis algorithm that given a safety property

$$
\begin{array}{rcl}
\langle\!\langle \big(\mathsf{tick}.\mathcal{A};(\mathcal{A}^{\leq 5} \cup \mathsf{open\_req}.\mathcal{B}_4^z)\big)^* \rangle\!\rangle^{\mathcal{P}} & \triangleq & \mathsf{X},\ \text{for } \mathsf{X} = \langle\!\langle\ \mathsf{tick}.\mathcal{A};(\mathcal{A}^{\leq 5} \cup \mathsf{open\_req}.\mathcal{B}_4^z)\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} \\
\langle\!\langle\ \mathsf{tick}.\mathcal{A};(\mathcal{A}^{\leq 5} \cup \mathsf{open\_req}.\mathcal{B}_4^z)\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \langle\!\langle\ \mathsf{tick}.\mathcal{A}\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{U}},\ \text{for } \mathsf{U} = \langle\!\langle\ \mathcal{A}^{\leq 5} \cup \mathsf{open\_req}.\mathcal{B}_4^z\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} \\
\langle\!\langle\ \mathsf{tick}.\mathcal{A}\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{U}} & \triangleq & \mathsf{V},\ \text{for } \mathsf{V} = {}^{\mathsf{tick}}\!/_{\mathsf{tick}}.\langle\!\langle\ \mathcal{A}\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{U}} + \sum_{\alpha \in \mathcal{P}\setminus\{\mathsf{tick},\mathsf{end}\}} \alpha/\tau.\mathsf{V} \\
\langle\!\langle\ \mathcal{A}\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{U}} & \triangleq & \mathsf{A},\ \text{for } \mathsf{A} = \sum_{\alpha \in \mathcal{A}} \alpha/\alpha.\langle\!\langle\ \epsilon\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{U}} + \sum_{\alpha \in \mathcal{P}\setminus(\mathcal{A}\cup\{\mathsf{tick},\mathsf{end}\})} \alpha/\tau.\mathsf{A} \\
\langle\!\langle\ \mathcal{A}^{\leq 5} \cup \mathsf{open\_req}.\mathcal{B}_4^z\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \mathsf{W},\ \text{for } \mathsf{W} = {}^{\mathsf{end}}\!/_{\mathsf{end}}.\langle\!\langle\ \epsilon\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} + \big(\sum_{\alpha \in \mathcal{A}} \alpha/\alpha.\langle\!\langle\ \mathcal{A}^{\leq 4}\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}}\big) + {}^{\mathsf{open\_req}}\!/_{\mathsf{open\_req}}.\langle\!\langle\ \mathcal{B}_5^z\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} \\
\langle\!\langle\ \mathcal{B}_h^k\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \mathsf{B}_h^k,\ \text{for } \mathsf{B}_h^k = {}^{\overline{\mathsf{close}}}\!/_{\overline{\mathsf{close}}}.\langle\!\langle\ (\mathcal{A}\setminus\{\overline{\mathsf{close}}\})^{\leq h-1}\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} + \sum_{\alpha \in \mathcal{A}\setminus\{\overline{\mathsf{close}}\}} \alpha/\alpha.\langle\!\langle\ \mathcal{B}_{h-1}^k\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} + \mathsf{R} \\
\mathsf{R} & \triangleq & {}^{\mathsf{end}}\!/_{\mathsf{end}}.\langle\!\langle\ \mathsf{tick}.\mathcal{B}_6^{k-1}\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} + \sum_{\alpha \in \mathcal{P}\setminus(\mathcal{A}\cup\{\mathsf{tick},\mathsf{end}\})} \alpha/\tau.\mathsf{B}_h^k \\
\langle\!\langle\ \mathsf{tick}.\mathcal{B}_h^k\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \mathsf{Y},\ \text{for } \mathsf{Y} = {}^{\mathsf{tick}}\!/_{\mathsf{tick}}.\langle\!\langle\ \mathcal{B}_h^k\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} + \sum_{\alpha \in \mathcal{P}\setminus\{\mathsf{tick},\mathsf{end}\}} \alpha/\tau.\langle\!\langle\ \mathsf{tick}.\mathcal{B}_h^k\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} \\
\langle\!\langle\ \mathcal{B}_0^k\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \mathsf{B}_0^k,\ \text{for } \mathsf{B}_0^k = {}^{\mathsf{end}}\!/_{\mathsf{end}}.\langle\!\langle\ \mathsf{tick}.\mathcal{B}_6^{k-1}\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} + \sum_{\alpha \in \mathcal{P}\setminus\{\mathsf{tick},\mathsf{end}\}} \alpha/\tau.\mathsf{B}_0^k \\
\langle\!\langle\ \mathcal{B}_0^1\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \mathsf{B}_0^1,\ \text{for } \mathsf{B}_0^1 = {}^{\overline{\mathsf{close}}}\!/_{\overline{\mathsf{close}}}.\langle\!\langle\ \mathsf{end}.\epsilon\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} + \sum_{\alpha \in \mathcal{P}\setminus\{\overline{\mathsf{close}},\mathsf{tick},\mathsf{end}\}} \alpha/\tau.\mathsf{B}_0^1 \\
\langle\!\langle\ \mathsf{end}.\epsilon\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \mathsf{Z},\ \text{for } \mathsf{Z} = {}^{\mathsf{end}}\!/_{\mathsf{end}}.\langle\!\langle\ \epsilon\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} + \sum_{\alpha \in \mathcal{P}\setminus\{\mathsf{tick},\mathsf{end}\}} \alpha/\tau.\mathsf{Z} \\
\langle\!\langle\ \epsilon\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \mathsf{X}
\end{array}
$$

TABLE V
SYNTHESIS FROM THE TIMED FORWARD CAUSALITY PROPERTY e_1 OF PLC_1, WHERE $\mathcal{P} = \{l_1, m_1, h_1\} \cup \{\overline{\mathsf{on}_1}, \overline{\mathsf{on}_2}, \overline{\mathsf{off}_1}, \overline{\mathsf{off}_2}, \overline{\mathsf{open}}, \overline{\mathsf{close}}\} \cup \{\mathsf{close\_req}, \mathsf{open\_req}\} \cup \{\mathsf{tick}, \mathsf{end}\}$, AND $\mathcal{A} = \{l_1, m_1, h_1\} \cup \{\overline{\mathsf{on}_1}, \overline{\mathsf{on}_2}, \overline{\mathsf{off}_1}, \overline{\mathsf{off}_2}, \overline{\mathsf{open}}, \overline{\mathsf{close}}\} \cup \{\mathsf{close\_req}\} \cup \{\mathsf{tick}\}$, FOR $k \in \{2,3,4\}$, AND $h \in \{1,2,3,4,5,6\}$.

$$
\begin{array}{rcl}
\langle\!\langle \big(\mathsf{tick}.\mathcal{A};(\overline{\mathsf{open\_req}}.\mathsf{end}.\mathcal{B}^2 \cup \overline{\mathsf{close\_req}}.\mathsf{end}.\mathcal{B}^2)\big)^* \rangle\!\rangle^{\mathcal{P}} & \triangleq & \mathsf{X},\ \text{for } \mathsf{X} = \langle\!\langle\ \mathsf{tick}.\mathcal{A};(\overline{\mathsf{open\_req}}.\mathsf{end}.\mathcal{B}^2 \cup \overline{\mathsf{close\_req}}.\mathsf{end}.\mathcal{B}^2)\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} \\
\langle\!\langle\ \mathsf{tick}.\mathcal{A};(\overline{\mathsf{open\_req}}.\mathsf{end}.\mathcal{B}^2 \cup \overline{\mathsf{close\_req}}.\mathsf{end}.\mathcal{B}^2)\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \langle\!\langle\ \mathsf{tick}.\mathcal{A}\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{U}},\ \text{for } \mathsf{U} = \langle\!\langle\ \overline{\mathsf{open\_req}}.\mathsf{end}.\mathcal{B}^2 \cup \overline{\mathsf{close\_req}}.\mathsf{end}.\mathcal{B}^2\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} \\
\langle\!\langle\ \mathsf{tick}.\mathcal{A}\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{U}} & \triangleq & \mathsf{V},\ \text{for } \mathsf{V} = {}^{\mathsf{tick}}\!/_{\mathsf{tick}}.\langle\!\langle\ \mathcal{A}\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{U}} + \sum_{\alpha \in \mathcal{P}\setminus\{\mathsf{tick},\mathsf{end}\}} \alpha/\tau.\mathsf{V} \\
\langle\!\langle\ \mathcal{A}\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{U}} & \triangleq & \mathsf{A},\ \text{for } \mathsf{A} = \sum_{\alpha \in \mathcal{A}} \alpha/\alpha.\langle\!\langle\ \epsilon\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{U}} + \sum_{\alpha \in \mathcal{P}\setminus(\mathcal{A}\cup\{\mathsf{tick},\mathsf{end}\})} \alpha/\tau.\mathsf{A} \\
\langle\!\langle\ \overline{\mathsf{open\_req}}.\mathsf{end}.\mathcal{B}^2 \cup \overline{\mathsf{close\_req}}.\mathsf{end}.\mathcal{B}^2\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \mathsf{W},\ \text{for } \mathsf{W} = {}^{\overline{\mathsf{open\_req}}}\!/_{\overline{\mathsf{open\_req}}}.\langle\!\langle\ \mathsf{end}.\mathcal{B}^2\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} + {}^{\overline{\mathsf{close\_req}}}\!/_{\overline{\mathsf{close\_req}}}.\langle\!\langle\ \mathsf{end}.\mathcal{B}^2\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} + \mathsf{R} \\
\mathsf{R} & \triangleq & \sum_{\alpha \in \mathcal{P}\setminus\{\overline{\mathsf{open\_req}},\overline{\mathsf{close\_req}},\mathsf{tick},\mathsf{end}\}} \alpha/\tau.\mathsf{W} \\
\langle\!\langle\ \mathsf{end}.\mathcal{B}^2\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \mathsf{B},\ \text{for } \mathsf{B} = {}^{\mathsf{end}}\!/_{\mathsf{end}}.\langle\!\langle\ \mathcal{B}^2\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} + \sum_{\alpha \in \mathcal{P}\setminus\{\mathsf{tick},\mathsf{end}\}} \alpha/\tau.\mathsf{B} \\
\langle\!\langle\ \mathcal{B}^k\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \mathsf{B}^k,\ \text{for } \mathsf{B}^k = {}^{\mathsf{tick}}\!/_{\mathsf{tick}}.\langle\!\langle\ \mathcal{A}^{\leq 2};\mathcal{B}^{k-1}\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} + \sum_{\alpha \in \mathcal{P}\setminus\{\mathsf{tick},\mathsf{end}\}} \alpha/\tau.\mathsf{B}^k \\
\langle\!\langle\ \mathcal{B}^0\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \mathsf{B}^0,\ \text{for } \mathsf{B}^0 = \langle\!\langle\ \epsilon\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} \\
\langle\!\langle\ \mathcal{A}^{\leq 2};\mathcal{B}^{k-1}\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \langle\!\langle\ \mathcal{A}^{\leq 2}\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{Y}},\ \text{for } \mathsf{Y} = \langle\!\langle\ \mathcal{B}^{k-1}\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} \\
\langle\!\langle\ \mathsf{end}.\epsilon\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \mathsf{Z},\ \text{for } \mathsf{Z} = {}^{\mathsf{end}}\!/_{\mathsf{end}}.\langle\!\langle\ \epsilon\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} + \sum_{\alpha \in \mathcal{P}\setminus\{\mathsf{tick},\mathsf{end}\}} \alpha/\tau.\mathsf{Z} \\
\langle\!\langle\ \epsilon\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \mathsf{X}
\end{array}
$$

TABLE VI
SYNTHESIS FROM THE TIMED BACKWARD CAUSALITY PROPERTY e_2 OF PLC_2, WHERE $\mathcal{P} = \{l_2, h_2\} \cup \{\mathsf{open\_req}, \mathsf{close\_req}\} \cup \{\mathsf{tick}, \mathsf{end}\}$, AND $\mathcal{A} = \{l_2, h_2\} \cup \{\mathsf{tick}\}$, $k \in \{1,2\}$.

$$
\begin{array}{rcl}
\langle\!\langle \big(\mathsf{tick}.\epsilon;(l_3.\mathcal{A}^{\leq 1} \cup h_3.\epsilon;\overline{\mathsf{on}_3}.\mathsf{end}.\epsilon)\big)^* \rangle\!\rangle^{\mathcal{P}} & \triangleq & \mathsf{X},\ \text{for } \mathsf{X} = \langle\!\langle\ \mathsf{tick}.\epsilon;(l_3.\mathcal{A}^{\leq 1} \cup h_3.\epsilon;\overline{\mathsf{on}_3}.\mathsf{end}.\epsilon)\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} \\
\langle\!\langle\ \mathsf{tick}.\epsilon;(l_3.\mathcal{A}^{\leq 1} \cup h_3.\epsilon;\overline{\mathsf{on}_3}.\mathsf{end}.\epsilon)\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \langle\!\langle\ \mathsf{tick}.\epsilon\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{U}},\ \text{for } \mathsf{U} = \langle\!\langle\ l_3.\mathcal{A}^{\leq 1} \cup h_3.\epsilon;\overline{\mathsf{on}_3}.\mathsf{end}.\epsilon\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} \\
\langle\!\langle\ \mathsf{tick}.\epsilon\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{U}} & \triangleq & \mathsf{V},\ \text{for } \mathsf{V} = {}^{\mathsf{tick}}\!/_{\mathsf{tick}}.\langle\!\langle\ \epsilon\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{U}} + \sum_{\alpha \in \mathcal{P}\setminus\{\mathsf{tick},\mathsf{end}\}} \alpha/\tau.\mathsf{V} \\
\langle\!\langle\ l_3.\mathcal{A}^{\leq 1} \cup h_3.\epsilon;\overline{\mathsf{on}_3}.\mathsf{end}.\epsilon\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \mathsf{W},\ \text{for } \mathsf{W} = {}^{l_3}\!/_{l_3}.\langle\!\langle\ \mathcal{A}^{\leq 1}\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} + {}^{h_3}\!/_{h_3}.\langle\!\langle\ \epsilon;\overline{\mathsf{on}_3}.\mathsf{end}.\epsilon\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} + \sum_{\alpha \in \mathcal{P}\setminus\{l_3,h_3,\mathsf{tick},\mathsf{end}\}} \alpha/\tau.\mathsf{W} \\
\langle\!\langle\ \epsilon;\overline{\mathsf{on}_3}.\mathsf{end}.\epsilon\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \langle\!\langle\ \epsilon\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{R}}\ \text{for } \mathsf{R} = \langle\!\langle\ \overline{\mathsf{on}_3}.\mathsf{end}.\epsilon\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} \\
\langle\!\langle\ \overline{\mathsf{on}_3}.\mathsf{end}.\epsilon\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \mathsf{Y},\ \text{for } \mathsf{Y} = {}^{\overline{\mathsf{on}_3}}\!/_{\overline{\mathsf{on}_3}}.\langle\!\langle\ \mathsf{end}.\epsilon\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} + \sum_{\alpha \in \mathcal{P}\setminus\{\overline{\mathsf{on}_3},\mathsf{tick},\mathsf{end}\}} \alpha/\tau.\mathsf{Y} \\
\langle\!\langle\ \mathsf{end}.\epsilon\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \mathsf{Z},\ \text{for } \mathsf{Z} = {}^{\mathsf{end}}\!/_{\mathsf{end}}.\langle\!\langle\ \epsilon\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} + \sum_{\alpha \in \mathcal{P}\setminus\{\mathsf{tick},\mathsf{end}\}} \alpha/\tau.\mathsf{Z} \\
\langle\!\langle\ \epsilon\ \rangle\!\rangle^{\mathcal{P}}_{\mathsf{X}} & \triangleq & \mathsf{X}
\end{array}
$$

TABLE VII
SYNTHESIS FROM THE TIMED BACKWARD CAUSALITY PROPERTY e_3 OF PLC_3, WHERE $\mathcal{P} = \{l_3, h_3\} \cup \{\overline{\mathsf{on}_3}, \overline{\mathsf{off}_3}\} \cup \{\mathsf{tick}, \mathsf{end}\}$, AND $\mathcal{A} = \{l_3, h_3\} \cup \{\overline{\mathsf{off}_3}\} \cup \{\mathsf{tick}\}$.

returns a monitor, called *shield*, that analyses both inputs and outputs of a reactive system, and enforces the desired property by correcting the minimum number of output actions. More recently, Pinisetty et al. [35] proposed a bi-directional runtime enforcement mechanism for reactive systems, and more generally for cyber-physical systems, to correct both inputs and outputs. They express the desired properties in terms of *Discrete Timed Automata* (DTA) whose labels are system actions. Thus, an execution trace satisfies a required property only if it ends up on a final state of the corresponding DTA. Although the authors do not identify specific classes of correctness properties as we aim to do, DTAs are obviously more expressive than our class of regular properties. However, as not all regular properties can be enforced [27], they proposed a more permissive enforcement mechanism that accepts also execution traces which may reach a final state.

Finally, Aceto et al. [26] developed an operational framework to enforce properties in HML logic with recursion (μHML) relying on suppression only. They also enforced the safety of the syntactic fragment of the logic by providing an automated synthesis algorithm that generates correct suppression monitors from formulas. Enforceability of modal μ-calculus (a reformulation of μHML) was previously tackled by Martinelli and Matteucci [36].

As regards papers in the context of *control system security* closer to our objectives, McLaughlin [10] proposed the introduction of an enforcement mechanism, called C^2, similar to our secure proxy, to mediate the control signals u_k transmitted by the PLC to the plant. Thus, like our secured proxy, C^2 is able to suppress commands, but unlike our proxy, it cannot autonomously send commands to the physical devices in the absence of a timely correct action from the PLC. Furthermore, C^2 does not seem to cope with inter-controller communications, and hence with colluding malware operating on PLCs of the same field network.

Mohan et al. [11] proposed a different approach by defining an ad-hoc security architecture, called *Secure System Simplex Architecture* (S3A), with the intention to generalise the notion of "correct system state" to include not just the physical state of the plant but also the *cyber state* of the PLCs of the system. In S3A, every PLC runs under the scrutiny of a *side-channel monitor* which looks for deviations with respect to *safe executions*, taking care of real-time constraints, memory usage, and communication patterns. If the information obtained via the monitor differs from the expected model(s) of the PLC, a *decision module* is informed to decide whether to pass the control from the "potentially compromised" PLC to a *safety controller* to maintain the plant within the required safety margins. As reported by the same authors, S3A has a number of limitations comprising: (i) the possible compromising of the side channels used for monitoring, (ii) the tuning of the timing parameters of the state machine, which is still a manual process.

Future work: We are currently working on the simulation of secure proxies, based on our enforcement mechanism, to monitor PLCs whose code is written in the *structured text* programming language, which is general enough to represent the other four languages for PLCs [37]. To this end, we have implemented in Python a synthesis algorithm that returns enforcers written in Verilog [38], an hardware description language used to model electronic systems. We are testing a number of case studies in a co-simulated environment, *i.e.*, an integration of two simulation environments built on top of Simulink [39] and ModelSim [40], where Simulink is used to run the system under scrutiny, while ModelSim simulates the runtime behaviour of the enforcer written in Verilog.

ACKNOWLEDGMENT

We thank the anonymous reviewers for their insightful comments. We also thank Adrian Francalanza, Yuan Gu and Davide Sangiorgi for their comments on early drafts of the paper. The authors have been partially supported by the project "Dipartimenti di Eccellenza 2018–2022" funded by the Italian Ministry of Education, Universities and Research (MIUR).

REFERENCES

[1] Y. Huang, A. A. Cárdenas, S. Amin, Z. Lin, H. Tsai, and S. Sastry, "Understanding the physical and economic consequences of attacks on control systems," *IJCIP*, vol. 2, no. 3, pp. 73–83, 2009.
[2] D. Kushner, "The real story of STUXnet," *IEEE Spectrum*, vol. 50, no. 3, pp. 48 – 53, 2013.
[3] ICS-CERT, "Cyber-Attack Against Ukrainian Critical Infrastructure," 2015, https://ics-cert.us- cert.gov/alerts/IR-ALERT-H-16-056-01.
[4] B. Johnson, D. Caban, M. Krotofil, D. Scali, N. Brubaker, and C. Glyer, "Attackers deploy ICS attack framework "TRITON" and cause operational disruption to critical infrastructure," *Threat Research Blog*, 2017.
[5] B. Radvanovsky, "Project shine: 1,000,000 internet-connected SCADA and ICS systems and counting," 2013, Tofino Security.
[6] J.-O. Malchow and J. Klick, "Sicherheit in vernetzten systemen: 21," 2014, dFN-Workshop. Paulsen, C., 2014, ch. Erreichbarkeit von digitalen Steuergeräten - in Lagebild, pp. C2-C19.
[7] R. Spenneberg, M. Brüggerman, and H. Schwartke, "PLC-Blaster: A Worm Living Solely in the PLC," in *Black Hat*, 2016, pp. 1–16.
[8] N. Govil, A. Agrawal, and N. O. Tippenhauer, "On Ladder Logic Bombs in Industrial Control Systems," in *SECPRE@ESORICS 2017*, ser. LNCS, vol. 10683. Springer, 2018, pp. 110–126.
[9] A. Abbasi and M. Hashemi, "Ghost in the PLC Designing an Undetectable programmable Logic Controller Rootkit via Pin Control Attack," in *Black Hat*, 2016, pp. 1–35.
[10] S. E. McLaughlin, "CPS: stateful policy enforcement for control system device usage," in *ACSAC*. ACM, 2013, pp. 109–118.
[11] S. Mohan, S. Bak, E. Betti, H. Yun, L. Sha, and M. Caccamo, "S3A: secure system simplex architecture for enhanced security and robustness of cyber-physical systems," in *HiCoNS*. ACM, 2013, pp. 65–74.
[12] F. B. Schneider, "Enforceable security policies," *ACM Transactions on Information and System Security*, vol. 3, no. 1, pp. 30–50, 2000.
[13] J. Ligatti, L. Bauer, and D. Walker, "Edit automata: enforcement mechanisms for run-time security policies," *International Journal of Information Security*, vol. 4, no. 1-2, pp. 2–16, 2005.
[14] Y. Falcone, L. Mounier, J. Fernandez, and J. Richier, "Runtime enforcement monitors: composition, synthesis, and enforcement abilities," *Formal Methods in System Design*, vol. 38, no. 3, pp. 223–262, 2011.
[15] A. Francalanza, L. Aceto, A. Achilleos, D. P. Attard, I. Cassar, D. Della Monica, and A. Ingólfsdóttir, "A Foundation for Runtime Monitoring," in *RV*, ser. LNCS, vol. 10548. Springer, 2017, pp. 8–29.
[16] Y. Falcone, K. Havelund, and G. Reger, "A Tutorial on Runtime Verification," in *EDSS*, ser. NATO Science for Peace and Security Series. IOS Press, 2013, vol. 34, pp. 141–175.
[17] M. Leucker and C. Schallhart, "A brief account of runtime verification," *Journal of Logic Programming*, vol. 78, no. 5, pp. 293–303, 2009.
[18] R. Lanotte and M. Merro, "A semantic theory of the Internet of Things," *Information and Computation*, vol. 259, no. 1, pp. 72–101, 2018.
[19] R. Lanotte, M. Merro, and S. Tini, "A Probabilistic Calculus of Cyber-Physical Systems," *Information and Computation*, 2020.

[20] M. Abadi, B. Blanchet, and C. Fournet, "The Applied Pi Calculus: Mobile Values, New Names, and Secure Communication," *Journal of the ACM*, vol. 65, no. 1, pp. 1:1–1:41, 2018.
[21] D. Macedonio and M. Merro, "A semantic analysis of key management protocols for wireless sensor networks," *Science of Computer Programming*, vol. 81, pp. 53–78, 2014.
[22] R. Lanotte, M. Merro, A. Munteanu, and Viganò, L., "A Formal Approach to Physics-based Attacks in Cyber-physical Systems," *ACM Transactions on Privacy and Security*, vol. 23, no. 1, pp. 3:1–3:41, 2020.
[23] M. Hennessy and T. Regan, "A process algebra for timed systems." *Information and Computation*, vol. 117, no. 2, pp. 221–239, 1995.
[24] E. Bartocci, J. V. Deshmukh, A. Donzé, G. E. Fainekos, O. Maler, D. Nickovic, and S. Sankaranarayanan, "Specification-Based Monitoring of Cyber-Physical Systems: A Survey on Theory, Tools and Applications," in *Lectures on Runtime Verification - Introductory and Advanced Topics*, ser. LNCS. Springer, 2018, vol. 10457, pp. 135–175.
[25] M. Hennessy and R. Milner, "Algebraic Laws for Nondeterminism and Concurrency," *Journal of the ACM*, vol. 32, no. 1, pp. 137–161, 1985.
[26] L. Aceto, I. Cassar, A. Francalanza, and A. Ingólfsdóttir, "On runtime enforcement via suppressions," in *CONCUR*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, pp. 34:1–34:17.
[27] D. Beauquier, J. Cohen, and R. Lanotte, "Security policies enforcement using finite and pushdown edit automata," *International Journal of Information Security*, vol. 12, no. 4, pp. 319–336, 2013.
[28] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, and S. Ö. Kjartansson, "On the Complexity of Determinizing Monitors," in *CIAA*, ser. LNCS, vol. 10329. Springer, 2017, pp. 1–13.
[29] A. P. Mathur and N. O. Tippenhauer, "SWaT: a water treatment testbed for research and training on ICS security," in *CySWater@CPSWeek*. IEEE Computer Society, 2016, pp. 31–36.
[30] J. Goh, S. Adepu, K. N. Junejo, and A. Mathur, "A Dataset to Support Research in the Design of Secure Water Treatment systems," in *CRITIS*, ser. LNCS, vol. 10242. Springer, 2017, pp. 88–99.
[31] M. Bielova, "A theory of constructive and predictable runtime enforcement mechanisms," Ph.D. dissertation, University of Trento, 2011.
[32] Y. Falcone, J.-C. Fernandez, and L. Mounier, "What can you verify and enforce at runtime?" *International Journal on Software Tools for Technology Transfer*, vol. 14, no. 3, pp. 349–382, 2012.
[33] Z. Manna and A. Pnueli, "A Hierarchy of Temporal Properties," Stanford University, Tech. Rep., 1987.
[34] B. Könighofer, M. Alshiekh, R. Bloem, L. Humphrey, R. Könighofer, U. Topcu, and C. Wang, "Shield synthesis," *Formal Methods in System Design*, vol. 51, no. 2, pp. 332–361, 2017.
[35] P. S. Pinisetty, S.and Roop, S. Smyth, N. Allen, S. Tripakis, and R. V. Hanxleden, "Runtime Enforcement of Cyber-Physical Systems," *ACM TECS*, vol. 16, no. 5s, pp. 178:1–178:25, 2017.
[36] F. Martinelli and I. Matteucci, "Through modeling to synthesis of security automata," *ENTCS*, vol. 179, pp. 31–46, 2007.
[37] D. Darvas, I. Majzik, and E. Blanco Vinuela, "Formal Verification of Safety of PLC Based Control Software," in *IFM*, ser. LNCS, vol. 9681. Springer, 2016, pp. 508–522.
[38] D. Thomas and P. Moorby, *The Verilog® Hardware Description Language*. Springer Science & Business Media, 2008.
[39] The MathWorks Inc., *Symbolic Math Toolbox*, Natick, Massachusetts, United States, 2019.
[40] Mentor Graphics, "Mentor Graphics ModelSim," 2014.

APPENDIX

A. Proofs of Section V

In order to prove the polynomial complexity of the synthesis algorithm, we provide a formal definition of size of both global and local properties. Intuitively, the size of a property is given by the number of operators occurring in it.

Definition 7: Let $\mathsf{dim}() : \mathbb{Pr}\oplus\mathbb{p}\mathbb{G} \cup \mathbb{Pr}\oplus\mathbb{p}\mathbb{L} \to \mathbb{N}$ be our property-size function, defined as follows:

$$\begin{array}{lcl} \mathsf{dim}(p^*) & \triangleq & 1 + \mathsf{dim}(p) \\ \mathsf{dim}(\epsilon) & \triangleq & 1 \\ \mathsf{dim}(p_1; p_2) & \triangleq & \mathsf{dim}(p_1) + \mathsf{dim}(p_2) + 1 \\ \mathsf{dim}(\bigcup_{i\in I} \alpha_i.p_i) & \triangleq & \mid I \mid + \sum_{i\in I} \mathsf{dim}(p_i). \end{array}$$

Let us prove Proposition 2 (Polynomial complexity).

Proof: Let $e = p^*$, for some $p \in \mathbb{Pr}\oplus\mathbb{p}\mathbb{L}$. We prove that the recursive structure of the function returning $\langle\!|p^*|\!\rangle^{\mathcal{P}}$ can be characterised in the following form: $T(m) = T(m-1) + n$, with $m = \mathsf{dim}(p)$ and n being the size of the set $\mathcal{P}$. The result follows because $T(m) = T(m-1) + n$ is $\mathcal{O}(m \cdot n)$.

As $\langle\!|p^*|\!\rangle^{\mathcal{P}} \triangleq \mathsf{X}$, for $\mathsf{X} = \langle\!| \, p \, |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, the proof is by case analysis on the structure of the local property p, by examining each synthesis step in which the synthesis function is processing $m = \mathsf{dim}(p)$ symbols. In what follows we identify: *(i)* how many symbols of p the synthesis functions processes, *(ii)* how many times the synthesis function calls itself, and *(iii)* how many computations performs in that step.

– Let $p \equiv \epsilon$. As $\mathsf{dim}(\epsilon) = 1$, it follows that $T(1) = 1$.

– Let $p \equiv p_1; p_2$. Let $m = \mathsf{dim}(p_1; p_2)$. By definition, the synthesis $\langle\!|p_1; p_2|\!\rangle^{\mathcal{P}}$ does not consume any symbol and calls itself on p_1 and p_2 with m_1 and m_2 symbols, respectively, with $m_1 + m_2 = m - 1$. Thus, we can characterise the recursive structure as: $T(m) = T(m_1) + T(m_2)$. Notice that the complexity of this recursive form is smaller than the complexity of $T(m-1) + n$.

– Let $p \equiv \bigcup_{i\in I} \pi_i.p_i$. Let $m = \mathsf{dim}(\bigcup_{i\in I} \pi_i.p_i)$. By definition, the synthesis $\langle\!|\bigcup_{i\in I} \pi_i.p_i|\!\rangle^{\mathcal{P}}$ consumes all events π_i, for $i \in I$. The synthesis algorithm re-calls itself $\mid I \mid$ times on p_i, with $\mathsf{dim}(p_i)$ symbols, for $i \in I$. Furthermore, the algorithm performs at most n operations due to a summation over over $\alpha \in \mathcal{P} \setminus (\bigcup_{i\in I} \pi_i \cup \{\mathsf{tick}, \mathsf{end}\})$, with $\mid \mathcal{P} \setminus (\bigcup_{i\in I} \pi_i \cup \{\mathsf{tick}, \mathsf{end}\}) \mid < n$. Thus, we can characterise the recursive structure as $T(m) = \sum_{i\in I} T(\mathsf{dim}(p_i)) + n$. Since $\sum_{i\in I} \mathsf{dim}(p_i) = m - \mid I \mid \leq m - 1$, the complexity is smaller than that of $T(m-1) + n$. ∎

Let us prove Theorem 1 (Transparency).

Proof: We prove a more general result. Let $e = p^*$ for $p \in \mathbb{Pr}\oplus\mathbb{p}\mathbb{L}$ and $P \in \mathbb{C}\mathbb{t}\mathbb{r}\mathbb{l}$. We prove that for any trace t of $\mathsf{go}\bowtie\{P\}$, if t is a prefix of some trace in $[\![p^*]\!]$ then:

1) $\langle\!|p^*|\!\rangle^{\mathcal{P}} \bowtie \{P\} \xrightarrow{t} \mathsf{E}\bowtie\{J\}$, where either $\mathsf{E} = \langle\!| \, p' \, |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$ or $\mathsf{E} = \mathsf{Z}$, with $\mathsf{Z} = \langle\!| \, p' \, |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, for some property p' sub-term of p, and for some automaton variable X and controller J;
2) there is a trace t' such that $t' \in [\![p']\!]$ and $t \cdot t'$ is a prefix of some trace in $[\![p^*]\!]$.

We proceed by induction on the length n of the trace t.

Base case. Let $n = 1$. Let $t = \beta \in \mathsf{Sens} \cup \mathsf{Chn}^* \cup \overline{\mathsf{Act}} \cup \{\mathsf{tick}, \mathsf{end}\}$ a trace of $\mathsf{go}\bowtie\{P\}$ which is a prefix of some trace in $[\![p^*]\!]$. By definition, $\beta \in \mathsf{events}(p^*) \subseteq \mathcal{P}$.

We now analyse the possible actions of the edit automaton $\langle\!|p^*|\!\rangle^{\mathcal{P}}$. By definition, $\langle\!|p^*|\!\rangle^{\mathcal{P}} \triangleq \mathsf{X}$, for $\mathsf{X} = \langle\!| \, p \, |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$. We proceed by case analysis on the structure of p:

– Let $p \equiv \epsilon$. This case is not admissible as p would not be well-formed.

– Let $p \equiv p_1; p_2$. By definition, the synthesis returns $\langle\!| \, p_1 \, |\!\rangle^{\mathcal{P}}_{\mathsf{Z}}$, for $\mathsf{Z} = \langle\!| \, p_2 \, |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, and $\mathsf{Z} \neq \mathsf{X}$. Now, if $p_1 \neq \epsilon$ then in order to analyse the transitions afforded by $\langle\!| \, p_1 \, |\!\rangle^{\mathcal{P}}_{\mathsf{Z}}$ we resort to one of the other two cases. Similarly, if $p_1 = \epsilon$ then $\langle\!| \, p_1 \, |\!\rangle^{\mathcal{P}}_{\mathsf{Z}} = \mathsf{Z}$, with $\mathsf{Z} = \langle\!| \, p_2 \, |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, and for the analysis of $\langle\!| \, p_2 \, |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$ we resort to one of the other cases.

– Let $p \equiv \bigcup_{i\in I} \pi_i.p_i$. By definition, the synthesis algorithm returns an edit automaton defined via the following equation:

$$\mathsf{Z} = \sum_{i\in I} {}^{\pi_i}\!/_{\pi_i}.\langle\!|\ p_i\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}} + \sum_{\alpha\in\mathcal{P}\setminus(\cup_{i\in I}\pi_i\cup\{\mathsf{tick},\mathsf{end}\})} {}^{\alpha}\!/_{\tau}.\mathsf{Z}\,.$$

Thus, the possible transitions of $\langle\!|\ p\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$ are:

- $\langle\!|\ p\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}} \xrightarrow{{}^{\pi_i}\!/_{\pi_i}} \langle\!|\ p_i\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, for $\pi_i \in \mathsf{events}(p^*)$;
- $\langle\!|\ p\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}} \xrightarrow{{}^{\alpha}\!/_{\tau}} \mathsf{Z}$, for $\alpha \in \mathcal{P}\setminus(\cup_{i\in I}\pi_i \cup \{\mathsf{tick},\mathsf{end}\})$.

Since β is a prefix of some trace in $[\![p^*]\!]$, then it follows that $\beta = \pi_j$, for some $j \in I$. By an application of rule (Enforce), triggered by rules (recE) and (Rec), we have that:

1) $\langle\!|p^*|\!\rangle^{\mathcal{P}} \bowtie \{P\} \xrightarrow{\beta} \langle\!|\ p_j\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}} \bowtie \{J\}$, and
2) since $[\![p_j]\!] \neq \emptyset$, there is a possibly empty trace $t' \in [\![p_j]\!]$ such that $\beta \cdot t'$ is a prefix of some trace in $[\![p^*]\!]$.

Inductive case. Let $n \in \mathbb{N}$, for some $n > 1$. Let t be a trace of $\mathsf{go}\bowtie\{P\}$ long n such that t is a prefix of some trace in $[\![p^*]\!]$. Since $n > 1$, then $t = t'' \cdot \beta$, for some trace t'' and action β. By inductive hypothesis we have:

1) $\langle\!|p^*|\!\rangle^{\mathcal{P}} \bowtie \{P\} \xrightarrow{t''} \mathsf{E} \bowtie \{J\}$, where either $\mathsf{E} = \langle\!|\ p'\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$ or $\mathsf{E} = \mathsf{Z}$, with $\mathsf{Z} = \langle\!|\ p'\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, for some property p' sub-term of p, and for some variable X and controller J;
2) there is a trace t' such that $t' \in [\![p']\!]$ and $t'' \cdot t'$ is a prefix of some trace in $[\![p^*]\!]$.

We now analyse the possible transitions of the edit automaton $\langle\!|\ p'\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$. We proceed by case analysis on the structure of the property p':

– Let $p' \equiv \epsilon$. By definition, the synthesis returns an automaton variable X defined via an equation. Thus, we resort to one of the other cases, depending on the definition of X.

– Let $p' \equiv p'_1;p'_2$. By definition, the synthesis returns $\langle\!|\ p'_1\ |\!\rangle^{\mathcal{P}}_{\mathsf{Z}}$, for $Z = \langle\!|\ p'_2\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, and $\mathsf{Z} \neq \mathsf{X}$. Thus, in order to analyse the possible transitions of $\langle\!|\ p'_1\ |\!\rangle^{\mathcal{P}}_{\mathsf{Z}}$ we resort to one of the other two cases.

– Let $p' \equiv \bigcup_{i\in I} \pi_i.p'_i$. By definition, the synthesis algorithm of Table III returns an edit automaton defined via the following recursive equation:

$$\mathsf{Z} = \sum_{i\in I} {}^{\pi_i}\!/_{\pi_i}.\langle\!|\ p'_i\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}} + \sum_{\alpha\in\mathcal{P}\setminus(\cup_{i\in I}\pi_i\cup\{\mathsf{tick},\mathsf{end}\})} {}^{\alpha}\!/_{\tau}.\mathsf{Z}\,.$$

Thus, the possible transitions of $\langle\!|\ p'\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$ are:

- $\langle\!|\ p'\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}} \xrightarrow{{}^{\pi_i}\!/_{\pi_i}} \langle\!|\ p'_i\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, for $\pi_i \in \mathsf{events}(p^*)$;
- $\langle\!|\ p'\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}} \xrightarrow{{}^{\alpha}\!/_{\tau}} \mathsf{Z}$, for $\alpha \in \mathcal{P}\setminus(\cup_{i\in I}\pi_i \cup \{\mathsf{tick},\mathsf{end}\})$.

We recall that $t = t'' \cdot \beta$ is a prefix of some trace in $[\![p^*]\!]$. Furthermore, by inductive hypothesis, there is a trace $t' \in [\![p']\!] = [\![\bigcup_{i\in I} \pi_i.p'_i]\!]$ such that $t'' \cdot t'$ is a prefix of some trace in $[\![p^*]\!]$. It follows that $\beta = \pi_j$, for some $j \in I$. Thus, by an application of rule (Enforce) we have:

1) $\langle\!|\ p'\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}} \bowtie \{J\} \xrightarrow{\beta} \langle\!|\ p'_j\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}} \bowtie \{J'\}$, for some J',
2) there is a trace $t'_j \in [\![p'_j]\!]$ such that $\beta \cdot t'_j$ is a prefix of some trace in $[\![p']\!]$.

Finally, for $t = t'' \cdot \beta$, we derive the required result:

1) $\langle\!|\ p^*\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}} \bowtie \{J\} \xrightarrow{t} \langle\!|\ p'_j\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}} \bowtie \{J'\}$, for some J', and
2) there is a trace $t'_j \in [\![p'_j]\!]$ such that $t \cdot t'_j$ is a prefix of some trace in $[\![p^*]\!]$.

■

In order to prove Theorem 2 we need two lemmata.

Lemma 1 (Soundness of the synthesis): Let $e = p^*$, for some $p \in \mathbb{P}\mathrm{r}\mathbb{o}\mathbb{p}\mathbb{L}$, and $\mathcal{P}$ be a set of actions such that $\mathsf{events}(e) \subseteq \mathcal{P}$. Let $\langle\!|e|\!\rangle^{\mathcal{P}} \xrightarrow{\alpha_1/\beta_1} \ldots \xrightarrow{\alpha_n/\beta_n} \mathsf{E}$ be an arbitrary execution trace of the synthesised automaton $\langle\!|e|\!\rangle^{\mathcal{P}}$. Then,

1) for $t = \beta_1 \cdot \ldots \cdot \beta_n$ the trace $\hat{t}$ is a prefix of some trace in $[\![e]\!]$, and
2) either $\mathsf{E} = \langle\!|\ p'\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, for some property p' sub-term of p and some automaton variable X, or $\mathsf{E} = \mathsf{Z}$, with $\mathsf{Z} = \langle\!|\ p''\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, for some property p'' sub-term of p, with p'' possibly equal to p, and some automaton variables X, Z.

Proof: By induction on the length of the execution trace.

Base case. Let $n = 1$. Let $\langle\!|p^*|\!\rangle^{\mathcal{P}} \xrightarrow{\alpha/\beta} \mathsf{E}$. As $\langle\!|p^*|\!\rangle^{\mathcal{P}} \triangleq \mathsf{X}$, for $\mathsf{X} = \langle\!|\ p\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, this transition may only be due to an application of rule (recE) because $\langle\!|\ p\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}} \xrightarrow{\alpha/\beta} \mathsf{E}$. Thus, we proceed by case analysis on the structure of the property p.

– Let $p \equiv \epsilon$. Impossible as p would not be well-formed.

– Let $p \equiv p_1;p_2$. By definition, the synthesis algorithm returns $\langle\!|\ p_1\ |\!\rangle^{\mathcal{P}}_{\mathsf{Z}}$, for $\mathsf{Z} = \langle\!|\ p_2\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, and $\mathsf{Z} \neq \mathsf{X}$. Now, if $p_1 \neq \epsilon$ then in order to analyse the transitions afforded by $\langle\!|\ p_1\ |\!\rangle^{\mathcal{P}}_{\mathsf{Z}}$ we resort to one of the other cases. Similarly, if $p_1 = \epsilon$ then $\langle\!|\ p_1\ |\!\rangle^{\mathcal{P}}_{\mathsf{Z}} = \mathsf{Z}$, with $\mathsf{Z} = \langle\!|\ p_2\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, and for the analysis of $\langle\!|\ p_2\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$ we resort to one of the other cases.

– Let $p \equiv \bigcup_{i\in I} \pi_i.p_i$. By definition, the synthesis algorithm of Table III returns and edit automaton defined via the following recursive equation:

$$\mathsf{Z} = \sum_{i\in I} {}^{\pi_i}\!/_{\pi_i}.\langle\!|\ p_i\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}} + \sum_{\alpha\in\mathcal{P}\setminus(\cup_{i\in I}\pi_i\cup\{\mathsf{tick},\mathsf{end}\})} {}^{\alpha}\!/_{\tau}.\mathsf{Z}\,.$$

Thus, the edit automaton $\langle\!|\ p\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$ admits the following two families of transitions:

- $\langle\!|\ p\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}} \xrightarrow{{}^{\pi_i}\!/_{\pi_i}} \langle\!|\ p_i\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, for $\pi_i \in \mathsf{events}(e) \subseteq \mathcal{P}$;
- $\langle\!|\ p\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}} \xrightarrow{{}^{\alpha}\!/_{\tau}} \mathsf{Z}$, for and $\alpha \in \mathcal{P}\setminus(\cup_{i\in I}\pi_i \cup \{\mathsf{tick},\mathsf{end}\})$.

In the first case, it is easy to see that: 1) π_i is a prefix of some trace in $[\![p^*]\!]$, for any $i \in I$, and 2) in the derivative $\langle\!|\ p_i\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, the property p_i is a sub-term of p. In the latter case, 1) $\hat{\tau} = \epsilon$ is a prefix of $[\![p^*]\!]$, and 2) $\mathsf{Z} = \langle\!|\ p\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$.

Inductive case. Let $n \in \mathbb{N}$, with $n > 1$.

Let $\langle\!|p^*|\!\rangle^{\mathcal{P}} \xrightarrow{\alpha_1/\beta_1} \ldots \xrightarrow{\alpha_{n-1}/\beta_{n-1}} \mathsf{E}' \xrightarrow{\alpha_n/\beta_n} \mathsf{E}$. By inductive hypothesis we have that:

1) for $t' = \beta_1 \cdot \ldots \cdot \beta_{n-1}$, the trace $\hat{t'}$ is a prefix of some trace in $[\![p^*]\!]$, and
2) either $\mathsf{E}' = \langle\!|\ p'\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, for some property p' sub-term of p and some automaton variable X, or $\mathsf{E}' = \mathsf{Z}$, with $\mathsf{Z} = \langle\!|\ p''\ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, for some property p'' sub-term of p, with p'' possibly equal to p, and some automaton variables X, Z.

Let us focus on the transition $\mathsf{E}' \xrightarrow{\alpha_n/\beta_n} \mathsf{E}$. With a reasoning similar to that of the base case, we derive that: 1) either $\widehat{\beta_n}$ is a prefix of some trace in $[\![p']\!]$ or $\widehat{\beta_n}$ is a prefix of some

trace in $[\![p'']\!]$, and 2) either $\mathsf{E} = \langle\!|\, p_1' \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, for some automaton variable X and some property p_1' sub-term of p', or $\mathsf{E} = \mathsf{Z}$, with $\mathsf{Z} = \langle\!|\, p_2' \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, for some variables X and Z and some property p_2' sub-term of p', or $\mathsf{E} = \langle\!|\, p_1'' \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, for some variable X and some property p_1'' sub-term of p'', or $\mathsf{E} = \mathsf{Z}$, with $\mathsf{Z} = \langle\!|\, p_2'' \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, for some variables X and Z and some property p_2'' sub-term of p''.

Thus, for $t = t' \cdot \beta_n$, we derive that: 1) the trace $\hat{t} = \hat{t'} \cdot \widehat{\beta_n}$ is a prefix of some trace in $[\![p^*]\!]$, and 2) either $\mathsf{E} = \langle\!|\, q \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, for some variable X and some property q sub-term of p, or $\mathsf{E} = \mathsf{Z}$, with $\mathsf{Z} = \langle\!|\, r \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, for some variables X and Z and some property r sub-term of p.

■

Lemma 2 (Trace decomposition): Let $e = p^*$, for some $p \in \mathbb{P}\mathrm{rop}\mathbb{L}$, $P \in \mathbb{C}\mathrm{tr}\mathbb{l}$ and $\mathcal{P}$ be the set of all possible actions of P such that $\mathsf{events}(e) \subseteq \mathcal{P}$. Then, for any execution trace $\langle\!|e|\!\rangle^{\mathcal{P}} \bowtie \{P\} \xrightarrow{\beta_1} \mathsf{E}_1 \bowtie \{J_1\} \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_n} \mathsf{E}_n \bowtie \{J_n\}$ holds:

1) $\langle\!|e|\!\rangle^{\mathcal{P}} \xrightarrow{\alpha_1/\beta_1} \mathsf{E}_1 \xrightarrow{\alpha_2/\beta_2} \ldots \xrightarrow{\alpha_n/\beta_n} \mathsf{E}_n$, with $\alpha_i \in \mathcal{P}$,
2) $J_0 = P$ and either $J_{i-1} \xrightarrow{\alpha_i} J_i$ or $J_i = J_{i-1}$, for $1 \leq i \leq n$.

Proof: By induction on the length n of the execution trace $\langle\!|e|\!\rangle^{\mathcal{P}} \bowtie \{P\} \xrightarrow{\beta_1} \mathsf{E}_1 \bowtie \{J_1\} \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_n} \mathsf{E}_n \bowtie \{J_n\}$.

Base case. Let $n = 1$. Let $\langle\!|e|\!\rangle^{\mathcal{P}} \bowtie \{P\} \xrightarrow{\beta} \mathsf{E} \bowtie \{J\}$. The following facts hold: *(i)* $P \equiv X$, for $X = \mathsf{tick}.W$, can only yield a tick-action by an application of rule (Rec); *(ii)* the synthesis function in Table III never returns an edit automaton that suppresses a tick-action; *(iii)* $\mathsf{events}(e) \subseteq \mathcal{P}$. From these facts and by an application of rule (Enforce), triggered by applications of rule (recE) and (Rec), we derive that: 1) $\langle\!|e|\!\rangle^{\mathcal{P}} \xrightarrow{\mathsf{tick}/\mathsf{tick}} \mathsf{E}$, and 2) $P \xrightarrow{\mathsf{tick}} W$, for $W \in \mathbb{S}\mathrm{leep}$.

Inductive case. Let $n > 1$. By inductive hypothesis we have:

1) $\langle\!|e|\!\rangle^{\mathcal{P}} \xrightarrow{\alpha_1/\beta_1} \mathsf{E}_1 \xrightarrow{\alpha_2/\beta_2} \ldots \xrightarrow{\alpha_{n-1}/\beta_{n-1}} \mathsf{E}_{n-1}$, $\alpha_i \in \mathcal{P}$,
2) $J_0 = P$ and either $J_{i-1} \xrightarrow{\alpha_i} J_i$ or $J_i = J_{i-1}$, for $1 \leq i \leq n$.

Consider the action $\mathsf{E}_{n-1} \bowtie \{J_{n-1}\} \xrightarrow{\beta_n} \mathsf{E}_n \bowtie \{J_n\}$. By an application of Lemma 1 we have that either $\mathsf{E}_{n-1} = \langle\!|\, p' \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, for some property p' sub-term of p, or $\mathsf{E}_{n-1} = \mathsf{Z}$ with $\mathsf{Z} = \langle\!|\, p'' \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}}$ for some property p'' sub-term of p, with p'' possibly equal to p, and some automaton variable X and Z. We proceed by case analysis on the structure of p' (the case analysis for p'' is similar):

– Let $p' \equiv \epsilon$. By definition, its synthesis returns X. Thus, we resort to one of the other cases.

– Let $p' \equiv p_1'; p_2'$. By definition, the synthesis returns $\langle\!|\, p_1' \,|\!\rangle^{\mathcal{P}}_{\mathsf{Z}}$, for $\mathsf{Z} = \langle\!|\, p_2' \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}}$ and $\mathsf{Z} \neq \mathsf{X}$. Thus, in order to analyse the transitions of $\langle\!|\, p_1 \,|\!\rangle^{\mathcal{P}}_{\mathsf{Z}}$ we resort to one of the other cases.

– Let $p' \equiv \bigcup_{i \in I} \pi_i.p_i'$. By definition, the synthesis of Table III returns an edit automaton defined via the following recursive equation:

$$\mathsf{Z} = \sum_{i \in I} \pi_i/\pi_i.\langle\!|\, p_i' \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}} + \sum_{\alpha \in \mathcal{P} \setminus (\cup_{i \in I} \pi_i \cup \{\mathsf{tick}, \mathsf{end}\})} \alpha/\tau.\mathsf{Z}$$

Thus, the possible transitions of $\langle\!|\, p' \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}}$ are:

- $\langle\!|\, p' \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}} \xrightarrow{\pi_i/\pi_i} \langle\!|\, p_i' \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, for $\pi_i \in \mathsf{events}(e) \subseteq \mathcal{P}$,
- $\langle\!|\, p' \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}} \xrightarrow{\alpha/\tau} \mathsf{Z}$, for $\alpha \in \mathcal{P} \setminus (\cup_{i \in I} \pi_i \cup \{\mathsf{tick}, \mathsf{end}\})$.

Let us analyse the possible transitions of J_{n-1}.

1) Let $J_{n-1} \xrightarrow{\pi_i} J_n$, for $\pi_i \in \mathsf{events}(e) \subseteq \mathcal{P}$. As $\langle\!|\, p' \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}} \xrightarrow{\pi_i/\pi_i} \langle\!|\, p_i' \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, by an application of the rule (Enforce) we have $\langle\!|\, p \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}} \bowtie \{J_{n-1}\} \xrightarrow{\pi_i} \langle\!|\, p_i' \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}} \bowtie \{J_n\}$, which concludes the proof of this case.

2) Let $J_{n-1} \xrightarrow{\mathsf{end}} J_n$, with $\pi_i \neq \mathsf{end}$ for all $i \in I$. By an application of rule (Mitigation), the edit automaton may produce autonomously the action π_i. Thus, we have that $\langle\!|\, p' \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}} \bowtie \{J_{n-1}\} \xrightarrow{\pi_i} \langle\!|\, p_i' \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}} \bowtie \{J_n\}$, with $J_n = J_{n-1}$, which concludes the proof of this case.

3) Let $J_{n-1} \xrightarrow{\alpha} J_n$, for $\alpha \in \mathcal{P} \setminus (\cup_{i \in I} \pi_i \cup \{\mathsf{tick}, \mathsf{end}\})$. Since $\langle\!|\, p' \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}} \xrightarrow{\alpha/\tau} \mathsf{Z}$, by an application of rule (Enforce), we have $\langle\!|\, p' \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}} \bowtie \{J_{n-1}\} \xrightarrow{\tau} \mathsf{Z} \bowtie \{J_n\}$, which concludes the proof of this case.

4) Let $J_{n-1} \xrightarrow{\mathsf{tick}} J_n$, with $\pi_i \neq \mathsf{tick}$ for all $i \in I$. As the monitor $\langle\!|\, p' \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}}$ does not allow tick-actions, the monitored controller $\langle\!|\, p' \,|\!\rangle^{\mathcal{P}}_{\mathsf{X}} \bowtie \{J_{n-1}\}$ may not perform any action.

■

Let us prove Theorem 2 (Soundness).

Proof: Let $t = \beta_1 \cdot \ldots \cdot \beta_n$ be an execution trace such that $\langle\!|e|\!\rangle^{\mathcal{P}} \bowtie \{P\} \xrightarrow{t} \mathsf{E} \bowtie \{J\}$, for some $\mathsf{E} \in \mathbb{E}\mathrm{dit}$ and some controller J. By an application of Lemma 2 there exist $\mathsf{E}_i \in \mathbb{E}\mathrm{dit}$ and $\alpha_i \in \mathcal{P}$, for $1 \leq i \leq n$, such that:

$$\langle\!|e|\!\rangle^{\mathcal{P}} \xrightarrow{\alpha_1/\beta_1} \mathsf{E}_1 \xrightarrow{\alpha_2/\beta_2} \ldots \xrightarrow{\alpha_n/\beta_n} \mathsf{E}_n = \mathsf{E}\,.$$

By Lemma 1 the trace $\hat{t}$ is a prefix of some trace in $[\![e]\!]$.

■

Let us prove Proposition 3 (Deadlock-freedom).

Proof: We prove that if $\langle\!|e|\!\rangle^{\mathcal{P}} \bowtie \{P\} \xrightarrow{t} \mathsf{E} \bowtie \{J\}$, for some $\mathsf{E} \in \mathbb{E}\mathrm{dit}$ and some J, then there exist β, E' and J' such that $\mathsf{E} \bowtie \{J\} \xrightarrow{\beta} \mathsf{E}' \bowtie \{J'\}$. We proceed by case analysis on the structure of the controller J.

– Let $J = \mathsf{tick}^{k-l}.S$, for $0 \leq l \leq k-1$. In this case, we have to show that the enforcer allows tick-actions, i.e., $\mathsf{E} \xrightarrow{\mathsf{tick}/\mathsf{tick}} \mathsf{E}'$, thus $\beta = \mathsf{tick}$. According to the syntax of our controllers, there are two different possibilities to reach the controller J: either $P \xrightarrow{\mathsf{tick}} \ldots \xrightarrow{\mathsf{tick}} J$ or $P \xrightarrow{t'} \mathsf{end}.X \xrightarrow{\mathsf{end}} P \xrightarrow{\mathsf{tick}} \ldots \xrightarrow{\mathsf{tick}} J$, for some trace t'. We focus on the latter case, as the former is simpler. By an application of Lemma 2 we can decompose the trace $\langle\!|e|\!\rangle^{\mathcal{P}} \bowtie \{P\} \xrightarrow{t} \mathsf{E} \bowtie \{J\}$ as

$$\langle\!|e|\!\rangle^{\mathcal{P}} \bowtie \{J_0\} \xrightarrow{\beta_1} \mathsf{E}_1 \bowtie \{J_1\} \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_n} \mathsf{E}_n \bowtie \{J_n\}$$

such that:

1) $\langle\!|e|\!\rangle^{\mathcal{P}} \xrightarrow{\alpha_1/\beta_1} \mathsf{E}_1 \xrightarrow{\alpha_2/\beta_2} \ldots \xrightarrow{\alpha_n/\beta_n} \mathsf{E}_n = \mathsf{E}$, $\alpha_i \in \mathcal{P}$
2) $J_0 = P$, $J_n = J$, either $J_{i-1} \xrightarrow{\alpha_i} J_i$ or $J_i = J_{i-1}$, for $1 \leq i \leq n$.

By inspection on the synthesis algorithm we know that the edit automata resulting from the synthesis never suppress tick-actions and end-actions. Thus, the execution trace of item 1) can be refined as follows: $\langle\!|e|\!\rangle^{\mathcal{P}} \xrightarrow{\alpha_1/\beta_1} \mathsf{E}_1 \xrightarrow{\alpha_2/\beta_2} \ldots \xrightarrow{\alpha_{n-l-1}/\beta_{n-l-1}} \mathsf{E}_{n-l-1} \xrightarrow{\mathsf{end}/\mathsf{end}} \mathsf{E}_{n-l} \xrightarrow{\mathsf{tick}/\mathsf{tick}} \ldots \xrightarrow{\mathsf{tick}/\mathsf{tick}} \mathsf{E}_n = \mathsf{E}$. By an application of Lemma 1, the trace $\hat{t}$ is a

prefix of some trace in $[\![e]\!]$; as e is k-sleeping, it follows that $t = t' \cdot \mathsf{end} \cdot \mathsf{tick}^l$, with $t' = \beta_1 \cdot ... \cdot \beta_{n-l-1}$. Still by an application of Lemma 1 we have that either $\mathsf{E} = \langle\!|\ p' \ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, for some property p' sub-term of property p, or $\mathsf{E} = \mathsf{Z}$ with $\mathsf{Z} = \langle\!|\ p'' \ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$ for some property p'' sub-term of p, with p'' possibly equal to p, and some automaton variables X and Z. Summarising, E comes from the synthesis of some property. By inspection on the synthesis algorithm we know that the edit automata resulting from the synthesis never deadlock, *i.e.*, $\mathsf{E} \xrightarrow{\alpha/\beta} \mathsf{E}'$, for some E', α and β. By Lemma 1 the trace $\hat{t} \cdot \hat{\beta}$ must be a prefix of some trace in $[\![e]\!]$. As e is k-sleeping and the edit automata resulting from the synthesis never suppress tick-actions, it holds that $\mathsf{E} \xrightarrow{\mathsf{tick}/\mathsf{tick}} \mathsf{E}'$. As $J = \mathsf{tick}^{h-l}.S \xrightarrow{\mathsf{tick}} \mathsf{tick}^{h-l-1}.S = J'$, we have that $\mathsf{E}\bowtie\{J\} \xrightarrow{\beta} \mathsf{E}'\bowtie\{J'\}$, for $\beta = \mathsf{tick}$, as required.

– Let $J \in \mathbb{S}\mathrm{ens} \cup \mathbb{C}\mathrm{omm} \cup \mathbb{A}\mathrm{ct}$. We have to show that for any α-action of the controller there is an α/β-action of the enforcer, for some β. By an application of Lemma 1, we have that either $\mathsf{E} = \langle\!|\ p' \ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, for some property p' sub-term of p and some automaton variable X, or $\mathsf{E} = \mathsf{Z}$, with $\mathsf{Z} = \langle\!|\ p'' \ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, for some property p'' sub-term of p, with p'' possibly equal to p, and some automaton variables X, Z. We proceed by case analysis on the structure of p' (the case for p'' is similar):

- Let $p' \equiv \epsilon$. By definition it is synthesised into X. Thus, we resort to one of the other cases.
- Let $p' \equiv p'_1; p'_2$. By definition, the synthesis algorithm returns $\langle\!|\ p'_1 \ |\!\rangle^{\mathcal{P}}_{\mathsf{Z}}$, for $\mathsf{Z} = \langle\!|\ p'_2 \ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$ and $\mathsf{Z} \neq \mathsf{X}$. Thus, we resort to one of the other cases.
- Let $p' \equiv \bigcup_{i\in I} \pi_i.p'_i$. By definition, the synthesis algorithm of Table III returns an edit automaton defined via the following recursive equation:

$$\mathsf{Z} = \sum_{i\in I} \pi_i/\pi_i.\langle\!|\ p'_i \ |\!\rangle^{\mathcal{P}}_{\mathsf{X}} + \sum_{\alpha\in\mathcal{A}\setminus(\cup_{i\in I}\pi_i\cup\{\mathsf{tick},\mathsf{end}\})} \alpha/\tau.\mathsf{Z}$$

Thus, the possible transitions of $\langle\!|\ p' \ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$ are:

– $\langle\!|\ p' \ |\!\rangle^{\mathcal{P}}_{\mathsf{X}} \xrightarrow{\pi_i/\pi_i} \langle\!|\ p'_i \ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, for $\pi_i \in \mathsf{events}(e) \subseteq \mathcal{P}$,

– $\langle\!|\ p' \ |\!\rangle^{\mathcal{P}}_{\mathsf{X}} \xrightarrow{\alpha/\tau} \mathsf{Z}$, for $\alpha \in \mathcal{P} \setminus (\cup_{i\in I}\pi_i \cup \{\mathsf{tick},\mathsf{end}\})$.

Now, let us consider the possible transitions of J.

1) Let $J \xrightarrow{\pi_i} J'$, for $\pi_i \in \mathsf{events}(e) \subseteq \mathcal{P}$. As we have $\langle\!|\ p' \ |\!\rangle^{\mathcal{P}}_{\mathsf{X}} \xrightarrow{\pi_i/\pi_i} \langle\!|\ p'_i \ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$, by an application of rule (Enforce) it follows that $\mathsf{E}\bowtie\{J\} \xrightarrow{\pi_i} \langle\!|\ p'_i \ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}\bowtie\{J'\}$, as required, for $\mathsf{E}' = \langle\!|\ p'_i \ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}$ and $\beta = \pi_i$.

2) Let $J \xrightarrow{\mathsf{end}} J'$, with $\pi_i \neq \mathsf{end}$ for all $i \in I$. By an application of rule (Mitigation), we have $\langle\!|\ p' \ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}\bowtie\{J\} \xrightarrow{\pi_i} \langle\!|\ p'_i \ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}\bowtie\{J\}$, for some $i \in I$, as required, for $\beta = \pi_i$.

3) Let $J \xrightarrow{\mathsf{tick}} J'$, with $\pi_i \neq \mathsf{tick}$ for all $i \in I$. As $J \in \mathbb{S}\mathrm{ens}\cup\mathbb{C}\mathrm{omm}\cup\mathbb{A}\mathrm{ct}$, this tick-action can be derived only by an application of one of the following transition rules: (TimeoutS), (TimeoutInC) and (TimeoutOutC). By inspection on these three rules, we derive that there is a J'' such that $J \xrightarrow{\alpha} J''$, for some $\alpha \in (\mathsf{Chn}^* \cup \mathsf{Sens}) \subseteq \mathcal{P}$. Now, depending on α, by an application of rule (Enforce) we derive that either $\langle\!|\ p' \ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}\bowtie\{J\} \xrightarrow{\pi_i} \langle\!|\ p'_i \ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}\bowtie\{J''\}$ or $\langle\!|\ p' \ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}\bowtie\{J\} \xrightarrow{\tau} \mathsf{Z}\bowtie\{J''\}$, as required.

4) Let $J \xrightarrow{\alpha} J'$, for $\alpha \in \mathcal{P}\setminus(\cup_{i\in I}\pi_i \cup \{\mathsf{tick},\mathsf{end}\})$. Since $\langle\!|\ p' \ |\!\rangle^{\mathcal{P}}_{\mathsf{X}} \xrightarrow{\alpha/\tau} \mathsf{Z}$, by an application of rule (Enforce) we have $\langle\!|\ p' \ |\!\rangle^{\mathcal{P}}_{\mathsf{X}}\bowtie\{J\} \xrightarrow{\tau} \mathsf{Z}\bowtie\{J'\}$, as required.

■

In order to prove Proposition 4 we need a definition.

Definition 8: Let $\mathrm{pre}() : \mathbb{C}\mathrm{trl}\cup\mathbb{S}\mathrm{leep}\cup\mathbb{S}\mathrm{ens}\cup\mathbb{C}\mathrm{omm}\cup\mathbb{A}\mathrm{ct} \to \mathbb{N}$ be a function that given a controller J returns an upper bound to the number of transitions that may be performed by J before an end-action. The definition follows:

$$\begin{array}{lcl}
\mathrm{pre}(X) & \triangleq & \mathrm{pre}(\mathsf{tick}.W), \text{ with } X = \mathsf{tick}.W \\
\mathrm{pre}(\mathsf{tick}.W) & \triangleq & 1 + \mathrm{pre}(W) \\
\mathrm{pre}(\lfloor\sum_{i\in I} s_i.S_i\rfloor S) & \triangleq & 1 + max(\mathrm{pre}(S), max_{i\in I}(\mathrm{pre}(S_i))) \\
\mathrm{pre}(\lfloor\sum_{i\in I} c_i.C_i\rfloor C) & \triangleq & 1 + max(\mathrm{pre}(C), max_{i\in I}(\mathrm{pre}(C_i))) \\
\mathrm{pre}(\lfloor\overline{c}.C_1\rfloor C_2) & \triangleq & 1 + max(\mathrm{pre}(C_1), \mathrm{pre}(C_2)) \\
\mathrm{pre}(\overline{a}.A) & \triangleq & 1 + \mathrm{pre}(A) \\
\mathrm{pre}(\mathsf{end}.X) & \triangleq & 0\,.
\end{array}$$

Let us prove Proposition 4 (Divergence-freedom).

Proof: Let $e = p^*$, for $p \in \mathbb{P}\mathrm{rop}\mathbb{L}$. Let $P \in \mathbb{C}\mathrm{trl}$, $\mathcal{P}$ be the set of all possible actions of P, and t a trace such that $\langle\!|e|\!\rangle^{\mathcal{P}} \bowtie \{P\} \xrightarrow{t} \mathsf{E}\bowtie\{J\}$. We define $k = \mathrm{pre}(P) + k_p$, where k_p is the length of the longest trace of $[\![p]\!]$. Notice that, by definition of our controllers, $\mathrm{pre}(P)$ is always finite. Furthermore, k_p is finite too as local properties in $\mathbb{P}\mathrm{rop}\mathbb{L}$ do not contain Kleene operators. As a consequence, k is finite. Thus, we prove that if $\mathsf{E}\bowtie\{J\} \xrightarrow{t'} \mathsf{E}'\bowtie\{J'\}$, with $\mid t' \mid \geq k$, then $\mathsf{end} \in t'$. More precisely, we prove that whenever $\mathsf{E}\bowtie\{J\} \xrightarrow{t'} \mathsf{E}'\bowtie\{J'\}$, with $\mid t' \mid \geq \mathrm{pre}(J) + k_p$, then $\mathsf{end} \in t'$. The result follows as J is a derivative of P, and hence $\mathrm{pre}(J) \leq \mathrm{pre}(P)$. We proceed by structural induction on J.

– Let $J \equiv \mathsf{end}.X$. Let $\mathsf{E}\bowtie\{J\} \xrightarrow{t'} \mathsf{E}'\bowtie\{J'\}$ such that $\mid t' \mid \geq \mathrm{pre}(\mathsf{end}.X) + k_p \geq k_p$. We reason by contradiction supposing that $\mathsf{end} \notin t'$. Since J may only perform an end-action, it follows that t' is entirely derived by applications of rule (Mitigation). In fact, rule (Enforce) cannot be used in the derivation of t' as $\mathsf{end} \notin t'$ and our synthesis never suppress end-actions. Notice that actions inferred by applications of rule (Mitigation) are always different from τ; thus, $\widehat{t'} = t'$ and $\widehat{t \cdot t'} = \widehat{t} \cdot t'$. By Theorem 2, the trace $\widehat{t \cdot t'}$ must be a prefix of some trace in $[\![p^*]\!]$. Since $\mid t' \mid \geq k_p$, where k_p is the length of the longest trace of $[\![p]\!]$, and p is well-formed, it follows that $\mathsf{end} \in t'$. In contradiction with the assumption $\mathsf{end} \notin t'$.

– Let $J \equiv \lfloor\sum_{i\in I} s_i.S_i\rfloor S$. Let $\mathsf{E}\bowtie\{J\} \xrightarrow{t'} \mathsf{E}'\bowtie\{J'\}$ such that $\mid t' \mid \geq \mathrm{pre}(J) + k_p$. Let β be the first action of t', i.e., $t' = \beta \cdot t''$, for some trace t'', such that $\mathsf{E}\bowtie\{J\} \xrightarrow{\beta} \mathsf{E}''\bowtie\{J''\} \xrightarrow{t''} \mathsf{E}'\bowtie\{J'\}$, for some E'' and J''. By inspection on J, it follows that either $J'' = S_i$, for some $i \in I$, or $J'' = S$. Since $\mid t' \mid \geq \mathrm{pre}(J) + k_p$, it follows that $\mid t'' \mid \geq \mathrm{pre}(J'') + k_p$. As J'' is sub-term of J, by inductive hypothesis it follows that $\mathsf{end} \in t''$. Thus, $\mathsf{end} \in t' = \beta \cdot t''$, as required.

The other cases are similar to the previous ones. ■