

# Incorporating Learning Modules to Improve the Resilient Design of Supervisory Cyber-Physical Systems

Prasanna Kannappan,<sup>1</sup> Konstantinos Karydis,<sup>2</sup> Herbert Tanner,<sup>1</sup> Adam Jardine,<sup>3</sup> and Jeffrey Heinz,<sup>3</sup>

**Abstract**—The paper contributes to the design of resilient supervisory Cyber-Physical Systems (CPSS) through the inclusion of appropriate learning modules in the subordinate autonomous agents. During normal operation, individual agents keep track of their supervisor’s commands and utilize the learning module, based on Grammatical Inference, to learn aspects of the organizational structure of the general system and role assignments. It is then shown that in cases that the supervisor fails or communication to subordinates is disrupted, these agents are able to recover normalcy of operations. Guaranteeing normalcy recovery in supervisory CPSS is critical in cases of a catastrophic failure or malicious attack.

**Keywords**—Cyber-Physical Systems, Resilience, Grammatical Inference, Formal Languages, Machine Learning.

## I. INTRODUCTION

A Cyber-Physical System (CPS) can include a wide range of hardware components, instrumented and controlled using a variety of sensors and actuators, and possibly networked via specific, sometimes dedicated, communication channels. Such system architectures are common in automated factories, smart and energy efficient buildings, the smart grid, and self-driving cars—among others [1], [2]—and are gradually finding application in national critical infrastructure systems, such as the water and electricity distribution grids. Spatially-distributed systems of this nature are often controlled centrally, but may become more vulnerable to failures or malicious attacks as the various system components become isolated from the central control authority. Thus, it is important to develop novel resilient design and control paradigms that go beyond the traditional notions of robustness, reliability, and stability [3].

<sup>1</sup>Department of Mechanical Engineering, University of Delaware, Newark, DE 19716 USA (e-mail: {prasanna, btanner}@udel.edu).

<sup>2</sup>Department of Mechanical Engineering & Applied Mechanics, University of Pennsylvania, Philadelphia, PA 19104 USA (e-mail: kkarydis@seas.upenn.edu).

<sup>3</sup>Department of Linguistics and Cognitive Science, University of Delaware, Newark, DE 19716 USA (e-mail: {ajardine, heinz}@udel.edu).

This work was supported by NSF CNS- at University of Delaware.

*Resilient design* [3] is a broad field of research that requires careful and creative integration of methodological elements borrowed from various—originally disjoint—control-theoretical areas with novel design and control paradigms. Examples of the former category include fault-tolerant control [4], resilient control [5], and robust control of networked systems [6], [7]. The latter category is a novel, rapidly expanding area of research, and some recent examples involve secure control [8], [9], and modeling of attacks and their impact [10], [11]. Recently-launched research initiatives such as FORCES [12] aim to increase the understanding of how to design resilient large-scale, human-in-the-loop CPSS by combining tools from resilient control with economic incentive schemes. Although resilience can emerge as a result of judicious design of interacting agent objectives and incentives, it is argued [3] that supervisory designs are still essential, much like any organization is unlikely to succeed without strong organizational leadership. At this time, it is unclear what happens in cases where the supervisor ceases functioning, following a malicious attack or a catastrophic failure. How can one prevent the whole system from collapsing *and* recover normalcy of operation?

To answer this question, we propose a different resilient design paradigm that incorporates learning to enable the system to recover normalcy of operation following a supervisor decapitation. We consider a class of supervisory CPSS, such as Supervisory Control and Data Acquisition (SCADA) systems, and integrate machine learning modules based on formal languages and *Grammatical Inference* [13] within the subordinate control units (agents). The idea is that agents keep track of the commands sent by the supervisor during normal operation, and through a particular machine learning procedure they can infer the plan(s) that the supervisor instructs them to execute, even if the agents themselves have no prior knowledge of those specific plans. We show that the agents can work together to ensure four key elements that prevent destabilization after supervisor decapitation: informa-

tion flow, consensus-reaching ability, functional capability, and information interpretation [14]. Finally, we show how to recover the language of the supervisor, and thus guarantee normalcy recovery.

Equipping the agents with a learning module allows them to learn aspects of the organizational patterns of a supervisory CPS, and role assignments during normal operation. Continuation of these elements following the supervisor decapitation has been found to be an important factor that contributes to resilience [15]. Additionally, the tools developed in this work are general, and can be used in a variety of applications such as emergency response, privacy, security and counter-intelligence, and learning through human-robot interactions.

## II. PROBLEM DESCRIPTION

The problem we consider is the following. We have  $k + 1$  agents, indexed in  $\{0, \dots, k\}$ , among which there is one designated coordinator—without loss of generality, we reserve the index 0 for this leader. In the present study we allow all-to-all communication.<sup>1</sup> The coordinator broadcasts instructions to all subordinates; each agent knows where to look in this broadcasted message to recover its own instruction plan.<sup>2</sup>

The group operates in an ambient environment, the state of which has conditional effects over which actions can be executed at any given environment state. In turn, the actions of the agents may or may not change the state of the environment. The dependency between agent actions and environment state is assumed Markovian, that is, actions that are compatible actions with a given world state depend only on the current state, and the next environment state depends entirely on what the agents do at the current state. It is assumed that the coordinator issues implementable plans, that is, the action commands it issues to the agents are all executable at the state where the agents are supposed to be.

The controlled (desired) behavior of agent  $i$  is captured by a formal language  $L_i$  over a particular alphabet of symbols,  $\Sigma_i$ . These terms are defined formally in Section III, but for now let us imagine the language of agent  $i$  as a set of finite combinations of letters, or symbols from  $\Sigma_i$ , which we will call words. We distinguish the *specification language* of agent  $i$   $L_i$ , which expresses what the agent should do, from the agent’s *capacity* which is a superset of  $L_i$  and expresses everything that this agent is capable of doing. The instructions

<sup>1</sup>If communication constraints exist, one may need to consider agents relaying information to one another over multiple hops.

<sup>2</sup>A plan is understood as a temporal sequence of actions.

given by the coordinator to agent  $i$  are definitely words that belong in  $L_i$ ; however, the plans that the coordinator issues also have to be consistent with the capacity of the agents. For that reason, the coordinator plans for agent  $i$  are naturally restricted to a subset of the specification language  $L_i$ . The latter is originally unknown to the agent.

What is known, however, is that the specification language belongs to a particular class of languages, and that this class is identifiable asymptotically from examples of elements from this language—we say that such a language is identifiable in the limit from positive presentations, and we define formally this notion of learning in Section III.

A first question now is whether an agent can learn its specification language if it observes the instructions given by its coordinator for sufficiently long time. The answer to this question is straightforward and is affirmative; this is a direct consequence of known results in the field of Grammatical Inference that deals with properties of classes of formal languages and associated learning techniques (more on that in Section III). What is not so clear is whether the agents can eventually collectively reconstruct *their coordinator* from the knowledge of their own specification language. The distinction is subtle, and it involves a certain type of synchronization that needs to occur between the strings in the individual specification languages—not every combination works. As it turns out, the answer to this question is also affirmative. The agents can reconstruct faithfully the function of their coordinator, should the latter seize to exist, through decentralized inference and inter-agent communication. Another implication of this assertion is that even if the coordinator had been trying to obscure its complete structure by delivering agent plans privately to each agent, there is a way for the agents to piece together their individual inferences and expose that obscured global structure.

## III. PRELIMINARIES

An *alphabet*  $\Sigma$  is a finite set of symbols. A *string*  $u$  is a finite concatenation of symbols in  $\Sigma$  of the form  $u = s_0 s_1 s_2 \cdots s_t$  such that  $s_i \in \Sigma$ . For a string  $w$  let  $|w|$  denote its length. The *empty string*  $\lambda$  is the string of length 0. For two strings  $u, v$ ,  $uv$  denotes their concatenation. Let  $\Sigma^*$  denote the set of all strings (including  $\lambda$ ) over the alphabet  $\Sigma$ . For strings  $v, w \in \Sigma^*$ ,  $v$  is a *substring* iff there exist some  $u_1, u_2 \in \Sigma^*$  such that  $u_1 v u_2 = w$ . The *k-factors* of a string  $w$ , denoted

$f_k(w)$ , are its substrings of length  $k$ . Formally,

$$f_k(w) = \begin{cases} \{u \in \Sigma^k \mid u \text{ is a substring of } w\} & \text{if } |w| \geq k \\ \{w\} & \text{otherwise} \end{cases}$$

Subsets of  $\Sigma^*$  are called *languages*. By default, all languages considered here are assumed to contain  $\lambda$ . A *grammar* is a finite representation of a (potentially infinite) language. It encodes in the form of rules, patterns that the symbols in the strings of the language should conform to. For a grammar  $G$ , let  $L(G)$  denote the language generated by  $G$ , that is, all the strings that can be formed by adhering to the rules of the grammar. A *class* of languages  $\mathcal{L}$  is a set of languages with some identifiable common characteristics.

This paper will make use of the class of languages which can be described by a grammar  $G$  that encodes the following simple rule: every string in the language should contain the same *single required  $k$ -factor*; i.e.,  $L(G) = \{w \mid G \in f_k(w)\}$ . We make use of this particular subclass because its member languages can be very efficiently learned from positive data, and it can be mapped to applications where a coordinator utilizes each subordinate so that it executes an agent-specific tactical maneuver at the right time. In this context, what the subordinate is trying to do is to understand what is that special maneuver that the coordinator makes it execute in every deployment.

The grammar rule just mentioned happens to characterize a particular subclass of a class of regular languages known as *Locally  $k$ -Testable*. *Locally  $k$ -Testable* languages are known to be identifiable in the limit [?], but the particular subclass considered admits very fast identification. In other words, if the agent specification is given by the automaton  $\mathcal{T}_i^L = \langle G_i, G_i^I, G_i^F, \Sigma_i, \rightarrow_i^L \rangle$ , and if  $z (z \in \Sigma_i^*, |z| = k)$  is the required substring of length  $k$ , then any string  $w$  that is accepted by the automaton  $\mathcal{T}_i^L$  is given by  $w = uzv$  where  $u, v \in \Sigma^*$ . For this particular subset of class of *Locally  $k$ -Testable* languages where each string  $w$  in language  $L$  is required to contain a particular substring  $z$ , the learner or **GIM** is defined as  $\text{GIM}(\phi[m]) \triangleq \bigcap_{r=1}^m f_k(\phi(r))$ , with  $f_k(\phi) \triangleq \{z \mid z \in \Sigma^k, |z| \geq k, \phi = uzv, u, v \in \Sigma^*\}$ , where  $\phi$  is a positive presentation of  $L$  and  $\phi[m]$  is a sequence of  $m$  positive presentations of  $L$ . Since we know that each word in language  $L$  is required to contain a specific substring  $z$ , the learner **GIM** would have converged when  $\text{GIM}(\phi[m]) = z$  or

$$|\text{GIM}(\phi[m])| = 1. \quad (1)$$

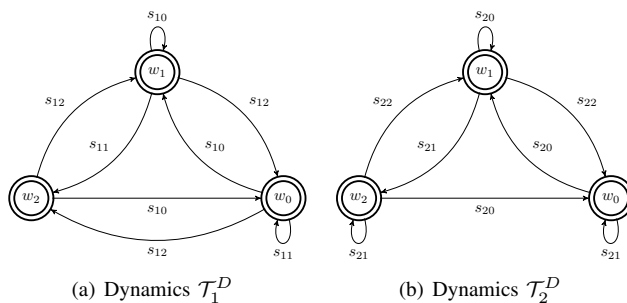


Fig. 1. The dynamics of agents  $\mathcal{T}_1$  and  $\mathcal{T}_2$  are shown in (a) and (b) respectively. All states here are both initial and final states of the transition systems. Note that all states here are bold double circles, i.e. they are both initial and final states.

## IV. METHODOLOGY

Consider two agents  $\mathcal{T}_1$  and  $\mathcal{T}_2$  with dynamics  $\mathcal{T}_1^D$  and  $\mathcal{T}_2^D$  that share the same set of states  $W$ , initial states  $W^I$  and final states  $W^F$ . The constrained dynamics can be specified as  $\mathcal{T}_1^C = \langle W \times G_1, W^I \times G_1^I, W^F \times G_1^F, \Sigma_1, \rightarrow_1^C \rangle$  and  $\mathcal{T}_2^C = \langle W \times G_2, W^I \times G_2^I, W^F \times G_2^F, \Sigma_2, \rightarrow_2^C \rangle$  respectively. Recall the Trim operation [16] on automata, which retains only their accessible<sup>3</sup> and co-accessible<sup>4</sup> states.

*Definition 1:* The synchronized product operation  $\otimes$  on the constrained dynamics'  $\mathcal{T}_1^C$  and  $\mathcal{T}_2^C$  is defined as

$$\begin{aligned} \mathcal{T}_{12}^S &= \mathcal{T}_1^C \otimes \mathcal{T}_2^C \\ &= \text{Trim}(\langle W \times G_1 \times G_2, W^I \times G_1^I \times G_2^I, \\ &\quad W^F \times G_1^F \times G_2^F, \Sigma_1 \times \Sigma_2, \rightarrow_{12}^S \rangle) \end{aligned} \quad (2)$$

where,  $\rightarrow_{12}^S: W \times G_1 \times G_2 \times \Sigma_1 \times \Sigma_2 \rightarrow W \times G_1 \times G_2$  is a transition function. For  $w \in W, g_1 \in G_1, g_2 \in G_2, \sigma_1 \in \Sigma_1$ , and  $\sigma_2 \in \Sigma_2$ ,  $\rightarrow_{12}^S(w, g_1, g_2, \sigma_1, \sigma_2) = (w', g_1', g_2')$  if  $\rightarrow_1^C(w, g_1, \sigma_1) = (w', g_1')$  and  $\rightarrow_2^C(w, g_2, \sigma_2) = (w', g_2')$ .

The synchronized product operation can be extended to more than two agents. For a case with  $n$  agents, the synchronized product operator  $\otimes$  is defined as  $\otimes: \mathcal{T}_1^C \times \mathcal{T}_2^C \times \dots \times \mathcal{T}_n^C \rightarrow \mathcal{T}_{12\dots n}^S$ .

### A. Case Study

Consider  $n$  agents  $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_n$  and a supervisor  $\mathcal{T}_0$ . In this framework, the agents receive sequence of actions or *plans* from the supervisor. During the initialization phase, i.e. before the supervisor starts sending plans to individual agents, we assume that an agent  $\mathcal{T}_i$  only knows its agent dynamics  $\mathcal{T}_i^D$ . The supervisor, with knowledge of all agent dynamics

<sup>3</sup>Accessible states are all states that are reachable from the initial states of a transition system.

<sup>4</sup>Co-accessible states are states from which there exists a path to a final state in the transition system.

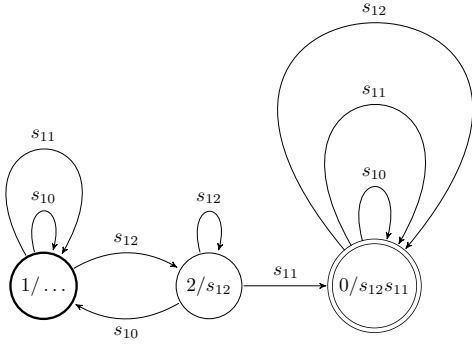


Fig. 2. Agent specification  $\mathcal{T}_1^L$  is shown here. The state labels shown are of the form  $j/z$  where  $j \in \mathbb{N} \cup \{0\}$  is an arbitrary number used to index a state and  $v$  is the satisfied part of the required substring  $z$ .

and specifications, computes the synchronization product of agents  $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_n$  to get the synchronization dynamics  $\mathcal{T}_{12\dots n}^S$  (2). We denote this synchronization dynamics  $\mathcal{T}_{12\dots n}^S$  and we identify it with supervisor dynamics  $\mathcal{T}_0$ .

Examples of two agent dynamics are depicted in Figs. 1(a) and 1(b). For this case study, it will be assumed that agent specifications belong to the subset of Locally-2-Testable class of languages that accepts strings that contain a required substring  $z$ . In this example, the agent  $\mathcal{T}_1$ 's specification  $\mathcal{T}_1^L$  accepts strings that contain the substring  $s_{12}s_{11}$ . The corresponding automaton for  $\mathcal{T}_1^L$  can be seen in Fig. 2. Similarly agent  $\mathcal{T}_2$ 's specification  $\mathcal{T}_2^L$  accepts strings that contain the substring  $s_{22}s_{21}$ . The states in the specification automaton of Fig. 2 are of the form  $j/z$  where  $j \in \mathbb{N} \cup \{0\}$  is an arbitrary number used to index the state and  $v$  is the satisfied part of the required substring  $z$ . The constrained dynamics  $\mathcal{T}_1^C$  that is computed by the supervisor for agent  $\mathcal{T}_1$  can be seen in Fig. 3. The synchronized dynamics of all agents or the supervisor specification  $\mathcal{T}_{12\dots n}^S$  can be seen in Fig. 4. After initialization phase, the supervisor uses its specification  $\mathcal{T}_{12\dots n}^S$  to generate plans for each agent. The plans  $\text{pl}$  are sequences of actions that the agents are to perform.

$$\begin{aligned} \text{pl} = & (w_1, g_{11}, g_{21}, \dots, g_{n1}) \xrightarrow{(s_{11}, s_{21}, \dots, s_{n1})} \\ & (w_2, g_{12}, g_{22}, \dots, g_{n2}) \xrightarrow{(s_{12}, s_{22}, \dots, s_{n2})} \dots \\ & \dots \xrightarrow{(s_{1p}, s_{2p}, \dots, s_{np})} (w_{p+1}, g_{1(p+1)}, \dots, g_{n(p+1)}) . \end{aligned} \quad (3)$$

Here  $i$  is the index of agent,  $n$  is the number of agents,  $r$  is the index of a plan,  $g_{ij} \in G_i$  is the component of agent specification  $\mathcal{T}_i^L$  corresponding to move  $j$  in a state tuple, and  $w_j \in W$  is the component of agent dynamics  $\mathcal{T}_i^D$  in a state tuple. Each element in path (3) is a node in the supervisor specification and each term on top of an arrow is the *action*

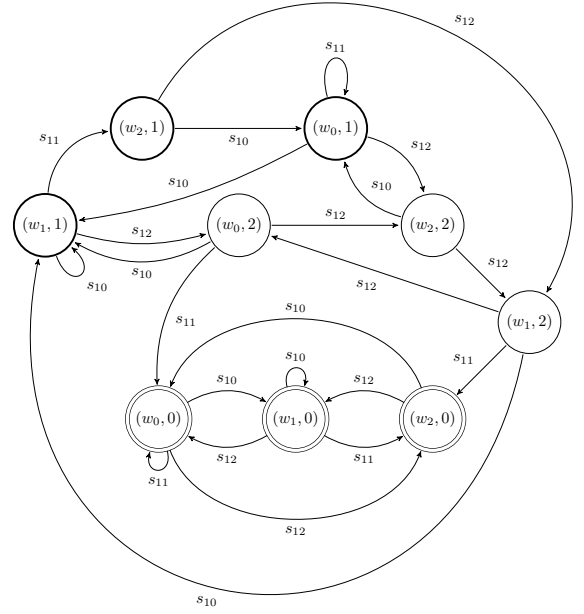


Fig. 3. Constrained dynamics  $\mathcal{T}_1^C$

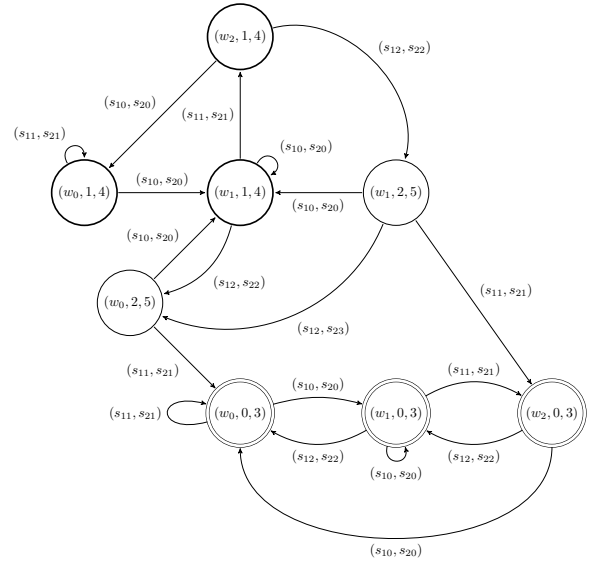


Fig. 4. Supervisor specification  $\mathcal{T}_{12\dots n}^S$

or edge the causes a transition between two nodes in the supervisor specification.

The plan  $\text{pl}$  generated by the supervisor can be expressed in form of a matrix as seen in (4).

$$\text{pl} \triangleq \begin{bmatrix} (w_1, g_{11}) & s_{11} & \dots & s_{1p} & (w_{p+1}, g_{1(p+1)}) \\ (w_1, g_{21}) & s_{21} & \dots & s_{2p} & (w_{p+1}, g_{2(p+1)}) \\ \dots & \dots & \ddots & \dots & \dots \\ (w_1, g_{n1}) & s_{n1} & \dots & s_{np} & (w_{p+1}, g_{n(p+1)}) \end{bmatrix} \quad (4)$$

Here each row corresponds to the plan for a single agent, i.e.,

row  $i$  comprises plan for agent  $\mathcal{T}_i$ . The supervisor controls agent  $\mathcal{T}_i$  by sending only row  $i$  of pl, denoted  $pl[[i]]$ .

It is a known result that each agent  $\mathcal{T}_i$  can learn its own specification language  $\mathcal{T}_i^L$  by observing positive presentations or in other words row  $i$  of a finite number of plans  $pl^1, pl^2, \dots, pl^m$  ( $m < \infty$ ) that the supervisor sends them. In this paper, we show that the agents  $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_n$  can also learn the supervisor specification  $\mathcal{T}_{12\dots n}^S$ .

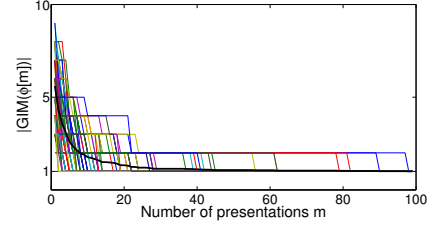
In order to learn the supervisor specification  $\mathcal{T}_{12\dots n}^S$ , each agent  $\mathcal{T}_i$  has to first learn its own specification  $\mathcal{T}_i^L$ . We know that the specification of agent  $\mathcal{T}_i$  belongs to a subset of Locally-k-Testable class of languages which requires all strings to contain a specific substring  $z_i$  of length  $k$ . Thus learning specification  $\mathcal{T}_i^L$  is reduced to learning  $z_i$ . Once each agent  $\mathcal{T}_i$  learns its required substring  $z_i$ , it can compute its specification  $\mathcal{T}_i^L$  and constrained dynamics  $\mathcal{T}_i^C$ . This constrained dynamics can then be broadcasted to all other agents in the group. At the end of these transmissions, all agents  $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_n$  know the constrained dynamics  $\mathcal{T}_1^C, \mathcal{T}_2^C, \dots, \mathcal{T}_n^C$  of all other agents. Then all agents  $\mathcal{T}_i$  can individually learn the supervisor's specification  $\mathcal{T}_{12\dots n}^S$  by computing the synchronized product of the constrained dynamics' (2). The steps involved in this learning procedure, specifically learning the required substring  $z_i$ , is discussed in the rest of this section.

Learning the required substring  $z_i$ , for an agent  $\mathcal{T}_i$ , is accomplished through grammatical inference. For instance if agent  $\mathcal{T}_i$  receives  $m$  plans  $pl^1[[i]], pl^2[[i]], \dots, pl^m[[i]]$  from the supervisor, then agent  $\mathcal{T}_i$  first extracts a sequence of presentations  $\phi[m]: \phi[m] \triangleq \phi_i(1)\phi_i(2)\dots\phi_i(m)$ , where  $\phi_i(r)$  is the largest subsequence of  $pl^r[[i]]$  that is a member of  $\Sigma_i^*$ . Agent  $\mathcal{T}_i$  learns the required substring  $z_i$  using GIM  $z_i = \text{GIM}(\phi[m])$ . When the number of presentations  $m$  available to an agent is sufficiently large enough to uniformly satisfy the convergence condition  $|\text{GIM}(\phi[m])| = 1$  for all agents, we can learn the required substring  $z_i$  for all agents.

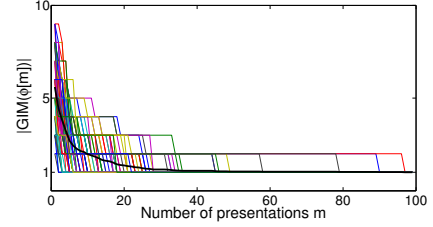
Once every  $z_i$  is learned, the procedure of computing the agent specification  $\mathcal{T}_i^L$ , agent constrained dynamics  $\mathcal{T}_i^C$  and the synchronized dynamics  $\mathcal{T}_{12\dots n}^S$  is essentially identical to the computations performed by the supervisor during initialization to produce  $\mathcal{T}_{12\dots n}^S$ . Learning  $z_i$  ultimately yields the supervisor specification  $\mathcal{T}_{12\dots n}^S$ .

## V. SIMULATIONS AND RESULTS

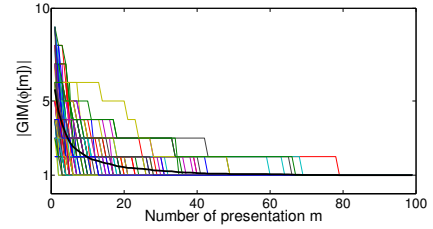
In this section we will experimentally show that the convergence condition (1) is uniformly satisfied for all agents,



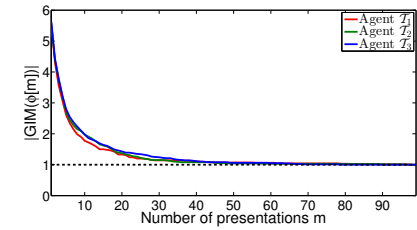
(a) Agent  $\mathcal{T}_1$



(b) Agent  $\mathcal{T}_2$



(c) Agent  $\mathcal{T}_3$



(d) All agents  $\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3$  means

Fig. 5. The convergence of  $|\text{GIM}(\phi[m])|$  to 1 as the number of presentations  $m$  increases for individual agents  $\mathcal{T}_1, \mathcal{T}_2$  and  $\mathcal{T}_3$  can be seen in Figures (a), (b) and (c) respectively. The colored lines in Figs. (a), (b) and (c) corresponds to plots for individual experiments and the solid black line corresponds to the mean value over all experiments. The mean solid black lines in Figs. (a), (b) and (c) are shown separately in Fig. (d). It is clear from Fig. (d) that the mean value of  $|\text{GIM}(\phi[m])|$  uniformly converges to 1 (shown as dotted black line) for all agents as the number of presentations  $m$  increases.

when the agents have observed a sufficiently large number of plans (i.e., when the number of plans  $m \rightarrow \infty$ ).

Consider a simulation environment with 3 agents  $\mathcal{T}_1, \mathcal{T}_2$  and  $\mathcal{T}_3$ , and a supervisor  $\mathcal{T}_0$ . The agent dynamics  $\mathcal{T}_i^D$  and the required substring  $z_i$  for each agent  $\mathcal{T}_i$  is randomly initialized with the following parameters: number of states = 3, minimum out degree of a node = 2, maximum out degree of a node = 3, number of alphabets per agent  $|\Sigma_i| = 3$ , and length of required substring  $k = 2$ . During the initialization phase,

each agent  $\mathcal{T}_i$  knows its dynamics, but does not know its required substring  $z_i$ . However each agent  $\mathcal{T}_i$  knows that the presentations it receives from the supervisor belongs to a subclass of Locally- $k$ -Testable ( $k = 2$ ) class of languages where each string contains the required substring  $z_i$ . On the other hand, the supervisor is privy to the dynamics ( $\mathcal{T}_1^D, \mathcal{T}_2^D$  and  $\mathcal{T}_3^D$ ) along with the required substrings ( $z_1, z_2$  and  $z_3$ ) of all agents. The supervisor uses this information to compute the agent specification  $\mathcal{T}_i^L$ , constrained dynamics  $\mathcal{T}_i^C$  and supervisor specification  $\mathcal{T}_{123}^S$ . The supervisor then uses the supervisor specification  $\mathcal{T}_{123}^S$  to generate  $m$  unique plans  $\text{pl}^r$  ( $r = 1, 2, \dots, m$ ,  $\text{pl}^{r_1} \neq \text{pl}^{r_2}$ , where  $r_1, r_2 \in r$ ) each of randomly chosen length  $p$  ( $k \leq p < 20$ ). The supervisor transmits row  $i$  (denoted  $\text{pl}^r[[i]]$ ) of plan  $\text{pl}^r$  to the respective agent  $\mathcal{T}_i$ . The agent  $\mathcal{T}_i$  uses the sequence of  $m$  plans to learn the required substring  $z_i$ .

To verify the result that the convergence condition is uniformly satisfied for all agents when  $m \rightarrow \infty$ , we run a series of 100 experiments and plot the results. For agent  $\mathcal{T}_1$ , the value of  $|\text{GIM}(\phi[m])|$  plotted against the number of presentations  $m$  for different experiments is shown as colored lines in Fig. 5(a). The solid black line in Fig. 5(a) is the mean value of  $|\text{GIM}(\phi[m])|$  over all 100 experiments. Similarly Figs. 5(b) and 5(c) show the corresponding plots for agents  $\mathcal{T}_2$  and  $\mathcal{T}_3$  respectively. The mean values of  $|\text{GIM}(\phi[m])|$  for all 3 agents over all experiments (shown as solid black lines in Figs. 5(a), 5(b) and 5(c)) are plotted together in Fig. 5(d). It is clear from Fig. 5(d), as the number of presentation  $m$  increases, the condition  $|\text{GIM}(\phi[m])| = 1$  (this reference value  $|\text{GIM}(\phi[m])| = 1$  is shown as a solid dotted line) is uniformly satisfied for all agents. Hence we have verified that by observing a large number of presentations  $m$ , we can successfully learn the supervisor specification  $\mathcal{T}_{12\dots n}^S$ .

This simulation was coded in python and run on a Intel-I7 Quad-core laptop computer (8 threads, 2.30GHz processor).

## VI. CONCLUSION

### REFERENCES

- [1] K.-D. Kim and P. Kumar, "Cyber-physical systems: A perspective at the centennial," *Proceedings of the IEEE*, vol. 100, no. Special Centennial Issue, pp. 1287–1308, 2012.
- [2] S. Khaitan and J. McCalley, "Design techniques and applications of cyberphysical systems: A survey," *IEEE Systems Journal*, vol. PP, no. 99, pp. 1–16, 2014.
- [3] C. Rieger, D. Gertman, and M. McQueen, "Resilient control systems: Next generation design research," in *2nd Conf. on Human System Interactions*, 2009, pp. 632–636.
- [4] M. Blanke, M. Kinnaert, J. Lunze, and M. Staroswiecki, *Diagnosis and fault-tolerant control*. Springer-Verlag, 2003.
- [5] M. S. Mahmoud, *Resilient Control of Uncertain Dynamical Systems*. Springer, 2004.
- [6] J. Hespanha, P. Naghshtabrizi, and Y. Xu, "A survey of recent results in networked control systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 138–162, 2007.
- [7] L. Schenato, B. Sinopoli, M. Franceschetti, K. Poolla, and S. Sastry, "Foundations of control and estimation over lossy networks," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 163–187, 2007.
- [8] A. Cardenas, S. Amin, and S. Sastry, "Secure control: Towards survivable cyber-physical systems," in *28th Int. Conf. on Distributed Computing Systems Workshops*, 2008, pp. 495–500.
- [9] Y. Mo and B. Sinopoli, "Secure control against replay attacks," in *47th Annual Allerton Conf. on Communication, Control, and Computing*, 2009, pp. 911–918.
- [10] D. Kundur, X. Feng, S. Mashayekh, S. Liu, T. Zourntos, and K. Butler-Purry, "Towards modelling the impact of cyber attacks on a smart grid," *International Journal of Security and Networks*, vol. 6, no. 1, pp. 2–13, 2011.
- [11] H. Cam, P. Moullem, Y. Mo, B. Sinopoli, and B. Nkrumah, "Modeling impact of attacks, recovery, and attackability conditions for situational awareness," in *IEEE Int. Inter-Disciplinary Conf. on Cognitive Methods in Situation Awareness and Decision Support*, 2014, pp. 181–187.
- [12] CPS-FORCES. (2015). [Online]. Available: <https://www.cps-forces.org/>
- [13] C. de la Higuera, *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, 2010.
- [14] J. Jordan, "When Heads Roll: Assessing the Effectiveness of Leadership Decapitation," *Security Studies*, vol. 18, no. 4, pp. 719–755, 2009.
- [15] J. M. Kendra and T. Wachtendorf, "Elements of Resilience After the World Trade Center Disaster: Reconstituting New York City's Emergency Operations Centre," *Disasters*, vol. 27, no. 1, pp. 37–53, 2003.
- [16] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Springer, 2008, vol. 11.