


# Norm Enforcement as Supervisory Control

Mehdi Dastani<sup>1</sup>, Sebastian Sardina<sup>2</sup>, and Vahid Yazdanpanah<sup>3</sup>

<sup>1</sup> Utrecht University, Utrecht, The Netherlands

`m.m.dastani@uu.nl`

<sup>2</sup> RMIT University, Melbourne, Australia

`sebastian.sardina@rmit.edu.au`

<sup>3</sup> University of Twente, Enschede, The Netherlands

`v.yazdanpanah@utwente.nl`

**Abstract.** In this paper, we study normative multi-agent systems from a supervisory control theory perspective. Concretely, we show how to model three well-known types of norm enforcement mechanisms by adopting well-studied supervisory control theory techniques for discrete event systems. Doing so provides a semantics for normative multi-agent systems rooted in formal languages and the ability to automatically synthesize SCT-based norm enforcement mechanisms for special, but still fairly expressive, type of systems and properties.

## 1 Introduction

In multi-agent systems literature, norms are proposed as a flexible means to regulate and control the behavior of autonomous agents in multi-agent settings [5]. Norms, for example, may indicate that certain states or actions are obligatory (obligation norms) or prohibited (prohibition norms) [15, 19]. There are various ways to enforce norms on a multi-agent system. For example, norms can be enforced by means of regimentation (i.e., by preventing violating behaviors), sanctioning (i.e., by allowing but sanctioning violating behaviors), or reparation (i.e., by allowing violating behaviors if some external repair event is expected). We emphasize that this paper merely focuses on regulative norms and dismiss constitutive norms [10]. Multi-agent systems that are controlled by means of norms are called *normative multi-agent systems*. In such systems, norms are often represented by logical formulas, which specify good or bad behaviors, while norm enforcement mechanisms are explained by means of model update, i.e., enforcing a norm on a multi-agent system is seen as updating the multi-agent model with the norm. Although these approaches contribute to the formal understanding and analysis of crucial concepts in normative multi-agent systems, they are less concerned with the implementability and complexity issues that are involved in synthesizing norm enforcement mechanisms.

In this paper, we consider norm enforcement as a “controllability” problem. Controlling autonomous processes has been the focus of extensive studies in *Supervisory Control Theory (SCT)* for (physical) Discrete Event Systems (DESS), with applications in a wide spectrum of (physical) systems, including manufacturing,

traffic, logistics, and communication systems [13]. The general goal in SCT is to control the system at hand by restricting its behavior as little as possible so that undesirable sequences of (discrete) events are prevented [21]. The significant advantage of SCT is its reliance on standard formal language theory, one of the most well known and accessible areas in Computer Science. This enables us to come-up with rigorous and implementable semantics for control mechanisms [9]. Indeed, the attractive computational properties in SCT have led to the development of various tools to synthesize control mechanisms (e.g., TCT/STCT [27], GRAIL [23], DESUMA [24], and SUPREMICA [20]).

In order to adopt techniques from SCT to synthesize norm-based enforcement mechanisms, we need to bridge the two mature fields of normative multi-agent systems and discrete event systems. To that end, we start by considering a multi-agent system as a plant and use in the rest of the paper the terms “plant” and “multi-agent systems” interchangeably. In our formalization, all possible behaviors of a plant can be generated by a finite state automaton; in case of a multi-agent system the automaton’s transitions represent joint actions. In SCT, the behavior of a plant is meant to be controlled—restricted—by a so-called *supervisor*. We assume perfect observability but limited control for plant supervisors. Plant supervisors have partial control in the sense that they can prevent/allow some but not all events. Using results and models from SCT, we propose regimentation, sanctioning, and repairing supervisors to model the corresponding forms of norm enforcement. We stress that it is *not* the objective of this work to claim that an SCT approach to norm modeling and reasoning is “better” than existing norm enforcement approaches. The novelty and significance of our contribution comes from connecting SCT and normative systems, two otherwise unrelated fields/problems, which has the potential of opening the door for synergies between the two.

The text is structured as follows. Section 2 provides some background on norms and norm enforcement mechanisms in multi-agent systems and Sect. 3 introduces a normative framework rooted in discrete event systems and supervisory control theory. Sections 4, 5, and 6 present formal models for regimentation, sanctioning, and repairing supervisors. Finally, Sects. 7 and 8 review related studies and conclude the paper.

## 2 Norms and Norm Enforcement

Imagine a computer system of a university that provides access to an intranet network. Students can log-in the computer system to browse material provided by the university. Via this account, students can also log-in to the Internet after which they are enabled to either watch some study-relevant tutorials or enjoy watching some study-irrelevant movies. The purpose of giving access to the Internet is to enable students to watch study-relevant tutorials, not to watch study-irrelevant movies. So, through the eyes of the system designer, it is not *normal* to watch movies using the provided Internet access. Any sequence of events that ends with a *movie watching* event (and all its extensions) is thus

seen as a *violating* behavior. For example, if a student logs into the university intranet and thereafter to the Internet, and then watches a movie followed by a tutorial, the behavior is considered as a violating one too. In our approach, a norm can be specified as a set of *event strings* that represent violating behaviors, i.e., a norm can be seen as a (possibly infinite) set of event sequences that are interpreted as “bad” behaviors. Note that this interpretation of norms is in line with the general idea that norms distinguish good and bad behaviors [8]. In our approach, a norm is specified by the bad behaviors; all other behaviors are considered to be good behaviors.

In order to avoid or suppress violating behaviors, various mechanisms have been proposed to enforce norms. For example, one may want to regiment norms in the sense that all violating behaviors are prevented. In our running example, regimentation would amount to blocking Internet access to students. This is based on the assumption that the designer of the system has control over students’ log-in access. Norm regimentation has some drawbacks, such as limiting the autonomy of students and preventing some compliant/good behaviors. For example, blocking the Internet access prevents students from watching study-relevant tutorials. An alternative approach would be to allow violating behaviors, but to impose *sanctions* on violating behaviors. In our scenario, this amounts to giving students free Internet access, but sanction the violating behaviors by closing the Internet access after some occurrences of watch-movie events, or to charge students the cost of the Internet access. In general, a sanction can either be modeled as an obligation for agents (in our case to oblige the student to pay the cost of Internet access) or forced by a controller/supervisor (in our case the supervisor withdraws money from the student’s deposit). The former type of sanctioning may result in multiple sanctioning rounds, as the agent may not comply to her obligations (e.g. to pay). In this paper, we follow the latter approach. Although norm sanctioning does not guarantee the prevention of norm-violating behaviors, it does not restrict the autonomy of agents nor prevents any norm-compliant behavior. Finally, a third approach is to allow norm violations if these are expected to be “repaired” by some other external events. In our scenario, students may be allowed to watch a movie if they watch a tutorial thereafter. This mechanism requires the possibility to predict the behavior of agents.

### 3 Preliminaries

In this section we develop a normative framework rooted in Discrete Event Systems (DESS) and Supervisory Control Theory (SCT) [13,21]. This will allow us to take advantage of the existing models and results from the respective communities and transfer them to norm-based multi-agent systems. Generally speaking, SCT is concerned with *imposing control* on the sequences of events (or strings/words) that such processes/systems -commonly referred to as the plant-may generate [21]. The techniques used by SCT are based on standard formal language theory [17].

We start by assuming a set of events  $\Sigma$  that can be generated by the multi-agent systems considered as the plant. A *language*  $L$  over the set of events  $\Sigma$  is

any set of finite sequences (strings/words) of events from  $\Sigma$ , i.e.,  $L \subseteq \Sigma^*$ . We use  $\epsilon \in \Sigma^*$  to denote the empty string. We assume the set of events to consist of disjoint sets of controllable  $\Sigma_c$  and uncontrollable events  $\Sigma_u$ , i.e.,  $\Sigma = \Sigma_c \cup \Sigma_u$ . The *prefix-closure* of a language  $L$ , denoted by  $\bar{L}$ , is the language of all prefixes of words in  $L$ , that is,  $w \in \bar{L}$  if and only if  $w.w' \in L$ , for some  $w' \in \Sigma^*$  ( $w.w'$  denotes the concatenation of words  $w$  and  $w'$ ). A language  $L$  is *prefix-closed* if  $L = \bar{L}$ .

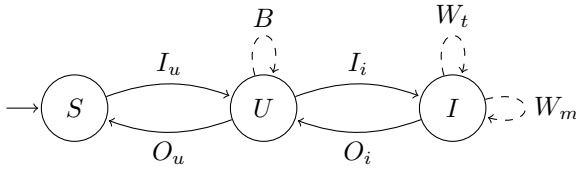
A multi-agent system is viewed then as a “generator” of the language of string of events. We note here that for multi-agent systems, an event may be an action profile in case we consider synchronized models or a single action of an individual agent in case of a turn-based multi-agent system. Formally, a *generator* is a deterministic finite-state machine  $\mathcal{G} = \langle \Sigma, G, g_0, \gamma, G_m \rangle$ , where  $\Sigma$  is the finite alphabet of events;  $G$  is a finite set of states;  $g_0 \in G$  is the initial state;  $\gamma : G \times \Sigma \mapsto G$  is the transition function; and  $G_m \subseteq G$  is the set of marked states. We generalize the transition function  $\gamma$  to words as follows:  $\gamma : G \times \Sigma^* \mapsto G$  is such that  $\gamma(g, \epsilon) = g$  and  $\gamma(g, w.\sigma) = \gamma(\gamma(g, w), \sigma)$ , with  $w \in \Sigma^*$  and  $\sigma \in \Sigma$ . We say that a state  $g \in G$  is *reachable* iff  $g = \gamma(g_0, w)$  for some word  $w \in \Sigma^*$ . Finally, given two words  $w_1, w_2 \in \Sigma^*$ ,  $w_1 \sqsubseteq w_2$  iff  $w_2 = w_1.w$  for some  $w \in \Sigma^*$ . Moreover,  $w_1 \sqsubset w_2$  iff  $w_2 = w_1.w$  for some  $w \in \Sigma^* \setminus \{\epsilon\}$ .

The *language* generated by generator  $\mathcal{G}$  is  $L(\mathcal{G}) = \{w \in \Sigma^* \mid \gamma(g_0, w) \text{ is defined}\}$ , where the *marked language* of  $\mathcal{G}$  is  $L_m(\mathcal{G}) = \{w \in L(\mathcal{G}) \mid \gamma(g_0, w) \in G_m\}$ . Words in the former language stand for, possibly partial, operations or tasks, while words in the marked language represent the completion of some operations or tasks. Note that  $L_m(\mathcal{G}) \subseteq L(\mathcal{G})$  and that  $L(\mathcal{G})$  is always prefix closed, while  $L_m(\mathcal{G})$  may not be.

A *norm*  $n$  is specified by a set of (finite) words, i.e.,  $n \subseteq \Sigma^*$ . An element of  $n$  is interpreted as a sequence of events that causes a violation. In case  $w_1, w_2 \in n$  such that  $w_1 \sqsubset w_2$ , we interpret  $w_2$  as a word causing more than one violation. Note that we take a semantic approach to norms and norm enforcement by focusing on agents’ (norm-compliant/-violating) behaviors. The language itself for describing such behaviors is out of the scope of this paper. The following example illustrates our formal framework by means of the running scenario.

**Example 1. Norm Enforcement Mechanisms.** Our running student scenario can be formally represented as  $\mathcal{S} = \langle \Sigma, G, g_0, \gamma \rangle$ , where  $\Sigma_c = \{I_u, I_i, O_u, O_i\}$  is the set of controllable events,  $\Sigma_u = \{W_t, W_m, B\}$  is the set of uncontrollable events, and  $\Sigma = \Sigma_u \cup \Sigma_c$ . The controllable events  $I_u$  and  $I_i$  stand for logging-in for *the University intranet* and *the Internet*, respectively;  $O_u$  and  $O_i$  stand for logging-out from *the university intranet* and *the Internet*, respectively. Moreover, the uncontrollable events  $W_t$  and  $W_m$  stand for *watching tutorial* and *watching movie*, respectively, and  $B$  for *browsing in the university intranet*. The set of states  $G$  consists of three possible states  $S$ ,  $U$ , and  $I$ , representing the states where (1) the student is neither logged-in for the university intranet nor for the Internet, (2) the student is logged-in for the university intranet and has access only to the provided university material, and (3) the student is logged-in for the Internet and has access to both tutorials and movies on the Internet. The initial

state of this plant is state  $S$  (i.e.,  $g_0 = \{S\}$ ) and the transition function  $\gamma$  is represented by the following graph.



In this scenario, we consider the norm of not watching a movie on the Internet to be specified by  $n = \Sigma^*.W_m$ . Note that all the members of norm  $n$  in this example end with a specific event  $W_m$ , suggesting that  $W_m$  is in fact the event that causes the violation. This is not the case in general as we do not use the concept of violating events in our framework. For instance, under an alternative norm  $n' = \Sigma^*.W_m e$ , with  $e \in \Sigma \setminus \{W_t\}$ , there is no single event causing the violation: watching a movie *and* not watching a tutorial immediately afterwards is the problem. While there are sequences of events containing  $W_m$  that are a case of violation (e.g.  $I_u I_i W_m O_i O_u$ ), there are also sequences of events containing  $W_m$  but that are not a case of violation (e.g.  $I_u I_i W_m W_t O_i O_u$ ). It follows then that, given a sequence of events  $\xi$ , verifying the occurrence of violation with respect to an arbitrary norm  $n$  is not, in general, reducible to syntactically checking whether  $\xi$  contains (or ends with) a specific class of violating events. This highlights our semantic approach to norms, as we focus on sequences of events (*system behaviors*) and see them as (potential) norm violations. Such view intrinsically contrasts to syntactic ones that are sensitive to the incidence of so called “violating events”. In further sections, we present a formal account of normative behaviors and illustrate how norm-violating/-compliant behaviors can be expressed in our SCT-based setting.

Given a multi-agent system and a norm specification, we aim at modeling the enforcement of the norm on the multi-agent system by means of a system supervisor. There are various forms of norm enforcement. In the following sections we model three well-known forms of norm enforcement in multi-agent systems using the concept of supervisory control in DESs. These three forms of norm enforcement are called regimentation, sanctioning and reparation. We say a supervisor enforces a norm on a multi-agent system by means of regimentation if norm violations are prevented. A norm is said to be enforced by means of sanctions if norm violations are allowed (not prevented), but compensated by some sanctions. Finally, a norm is said to be enforced by means of reparation if norm violations are followed by some reparation events generated by the multi-agent system itself. Norm sanctioning and norm reparation are similar in the sense that the system supervisor allows norms to be violated. However, they differ as norm sanctioning responds to norm violations by adding an external sanction event to repair the violations, while the reparation mechanism considers some system events as reparation events and allows norm violations when they are followed by reparation events. These forms of norm enforcement will be formally modeled in the following three sections.

## 4 Regiment-Based Supervision

In order to enforce a norm  $n \subseteq \Sigma^*$  on a multi-agent system by means of regimentation, we first identify norm violating and norm compliant behaviors under the regimentation mechanism. Given a norm specification containing all words that causes violations, the set of all “bad/undesired” behaviors (norm-violating behaviors) under regimentation consists of all words that simply extend a word from the norm specification. The set of “good/desired” behaviors (norm-compliant behaviors) under regimentation consists of all other behaviors.

**Definition 1. Norm Violating and Compliant behaviors.** *Let  $n \subseteq \Sigma^*$  be a norm. The set of  $n$ -violating behaviors under regimentation, denoted as  $Viol_n$ , is the set of suffixes of  $n$ , i.e.,  $Viol_n = n.\Sigma^*$ . The set of  $n$ -compliant behaviors under regimentation, denoted as  $K_n$ , is the complement of  $n$ -violating behaviors, i.e.,  $K_n = \Sigma^* \setminus Viol_n$ .*

In our running example,  $I_uBB, I_uBBI_i, I_uI_iW_t \in K_n$ . It should be clear that the concepts of norm specification, norm violating, and norm compliant behaviors are defined independent of a plant  $\mathcal{G}$ . Moreover, the set of norm violating behaviors and the set of norm compliant behaviors depend on the form of norm enforcement that we consider, which is regimentation in this case. Note that  $K_n$  is prefix-closed, i.e., if  $w.w' \in K_n$  then  $w \in K_n$ .

Since events that are involved in a system can be uncontrollable, it may not always be possible for the system supervisor to regiment a norm in order to prevent norm violating behaviors. For example, consider uncontrollable event  $u \in \Sigma_u$  and suppose  $u \in n$ . If the system generates a behavior that starts with event  $u$ , then the system supervisor cannot regiment the norm specified by  $n$  since  $u$  is an uncontrollable event. Therefore, given a system we identify which norms can be regimented by the system supervisor. A norm is said to be regimentable on a system if the occurrences of uncontrollable events in the system directly after  $n$ -compliant behaviors are  $n$ -compliant as well.

**Definition 2. Regimentability.** *A norm specified by  $n$  is regimentable in  $\mathcal{G}$  if  $\overline{K_n}.\Sigma_u \cap L(\mathcal{G}) \subseteq \overline{K_n}$ .*

Since  $K_n$  is prefix-closed, i.e.,  $\overline{K_n} = K_n$ , norm  $n$  is regimentable if  $K_n.\Sigma_u \cap L(\mathcal{G}) \subseteq K_n$ . Note that every norm is regimentable when the system does not involve uncontrollable events, i.e.,  $\Sigma_u = \emptyset$ . Also observe that the norm of our running example is *not* regimentable since  $I_uI_iW_t \in K_n$ , but  $I_uI_iW_tW_m \notin K_n$ . The non-regimentability of norms does not mean that bad behaviors cannot be prevented, it just states that not all good behaviors (which are also specified by norms) can be guaranteed if all bad behaviors are to be prevented. This is because, technically, the regimentability of a norm  $n$  is defined in terms of the good behaviors  $K_n$ . So, as we have seen, regimentability is, in some sense, a very demanding property. Then, given a norm  $n$ , one will generally be interested in the *largest subset* of good behaviors of  $K_n$  that can be allowed, if all bad behaviors are prevented. The following definition captures this.

**Definition 3. Supremal Regimentability.** Let  $\mathcal{G}$  be a multi-agent system and  $n \subseteq \Sigma^*$  a norm such that  $K_n \neq \emptyset$ . The supremally regimentable  $n$ -compliant behavior in  $\mathcal{G}$  is defined as  $K_n^\uparrow = \bigcup_{K \in R(K_n)} K$  where  $R(K_n) = \{K \subseteq K_n : \overline{K}.\Sigma_u \cap L(\mathcal{G}) \subseteq \overline{K}\}$ .

That is,  $K_n^\uparrow$  represents the largest sublanguage of  $K_n$  that satisfies the regimentability condition (Definition 2). Clearly,  $K_n^\uparrow$  only includes good behaviors, but maybe not all: some have to be sacrificed so as to guarantee the norm. Obviously, if the norm is (perfectly) regimentable, then the supremal is the whole  $K_n$ . Importantly, the supremal for a given norm is unique.

**Corollary 1.** For any norm  $n \subseteq \Sigma^*$ , we have that  $K_n^\uparrow$  is unique. Moreover, if  $n$  is regimentable, then  $K_n^\uparrow = K_n$ .

Next, we discuss the concrete mechanism used to regiment a multi-agent system so that violating behaviors are prevented. For this, we define a so-called *regimentation-based supervisor*, which intuitively speaking, controls a system by *enabling/disabling* controllable events at each execution moment.

**Definition 4. Regiment-Based Supervisor.** A regiment-based supervisor for a multi-agent system  $\mathcal{G}$  is a function of the form  $V_r : L(\mathcal{G}) \mapsto \{\Sigma_e \mid \Sigma_e \in 2^\Sigma, \Sigma_u \subseteq \Sigma_e\}$ , where  $V_r(w)$  denotes the set of events that are enabled (i.e., allowed) next.

Observe that supervisors must enable all uncontrollable events (i.e.,  $\Sigma_u \subseteq \Sigma_e$ )—they cannot be disabled. However, a supervisor may decide to disable—block—some controllable events (i.e., events in  $\Sigma_c$ ). The following definition captures what it means to supervise a multi-agent system.

**Definition 5. Regimentation-Based Supervision.** Let  $\mathcal{G}$  be a multi-agent system and  $V_r$  a regimentation-based supervisor for  $\mathcal{G}$ . The supervised language of  $\mathcal{G}$  under  $V_r$  is defined as  $L(V_r/\mathcal{G}) = \{w.\sigma \in L(\mathcal{G}) \mid w \in L(V_r/\mathcal{G}), \sigma \in V_r(w)\} \cup \{\epsilon\}$ .

That is,  $L(V_r/\mathcal{G})$  represents all behaviors that the multi-agent system  $\mathcal{G}$  may yield when supervised by  $V_r$ . Note that while recursively defined, the set  $L(V_r/\mathcal{G})$  is well-defined. Importing results from classical controllability [21], this supervision is a sufficient mechanism for regimentability of norms (if at all possible).

**Theorem 1.** Let  $\mathcal{G}$  be a multi-agent system and  $n \subseteq \Sigma^*$  a norm such that  $K_n \neq \emptyset$ . There exists a regiment-based norm supervisor  $V_r$  such that  $L(V_r/\mathcal{G}) = \overline{K_n}$  iff norm  $n$  is regimentable in  $\mathcal{G}$ .

*Proof.* We directly import the Controllability Theorem (CT) as presented in [13] (Page 145). According to CT, having a plant  $G$  that generates  $L(G)$  and a nonempty  $K \subseteq L(G)$ , there exists a supervisor  $S$  such that  $L(S/G) = \overline{K}$  iff  $\overline{K}.\Sigma_u \cap L(G) \subseteq \overline{K}$ . By considering our Definition 2 (Regimentability), we have Theorem 1.

For our running example, this theorem implies the non-existence of regimentation-based norm supervisor: norm  $n = \Sigma^*.W_m$  is *not* (perfectly) regimentable. Indeed, any supervisor should disable controllable event  $I_i$  in state  $U$  so as to prevent a behavior involving non-controllable event  $W_m$  happening (thus producing a norm violating behavior). However, doing so, will inevitably exclude norm-compliant behaviors that choose  $W_t$  in state  $I$ . Nonetheless, the next proposition shows that for any norm and plant, regardless of its regimentability, there exists a regiment-based supervisor that guarantees the supremally regimentable  $n$ -compliant behavior.

**Proposition 1.** *Let  $\mathcal{G}$  be a multi-agent system and  $n \subseteq \Sigma^*$  a norm such that  $K_n \neq \emptyset$ . Then, there exists a regiment-based supervisor  $V_r^*$  such that  $L(V_r^*/\mathcal{G}) = K_n^\uparrow$ .*

*Proof.*  $K_n^\uparrow$  satisfies the regimentability condition. The rest follows the proof of Theorem 1.

For our running example, this proposition ensures that there exists indeed a supervisor that can prevent all norm violating behaviors. Importantly, such a supervisor is *maximally permissive*: it is not possible to cater for more “good” behaviors without running the risk of violating some norm. An important result from SCT is that when both the plant and the specification are regular languages, and hence representable via finite automata (i.e., generators), the supervisor realizing the supremal realizable language can be finitely represented and in fact computed in polynomial time (w.r.t. the automata for the plant and specification) [26]. We can import such results as follows.

**Proposition 2.** *Let  $\mathcal{G}$  be a multi-agent system and  $n \subseteq \Sigma^*$  a norm for which there exists a generator  $\mathcal{G}_n$  such that  $L(\mathcal{G}_n) = n$ . Suppose further that  $K_n \neq \emptyset$ . Then, a generator  $\mathcal{R}$  such that  $L(\mathcal{R}) = K_n^\uparrow$  can be computed in polynomial time w.r.t.  $\mathcal{G}$  and  $\mathcal{G}_n$ .*

*Proof.* Because norm specification  $n$  is regular and implemented with  $\mathcal{G}_n$ , languages  $Viol_n$  and  $K_n$  are regular as well (and implementable with a generator that has one more state than  $\mathcal{G}_n$ ). Since  $\mathcal{G}$  is a generator, we can directly apply the results in [26] and the thesis follows.

We close by pointing out that besides leveraging on the solid theoretical foundations of SCT, the development above allows us to apply the existing tools for supervisor synthesis, such as TCT/STCT [27], GRAIL [23], DESUMA [24], and SUPREMICA [20], to automatically compute norm regimentation policies.

## 5 Sanction-Based Supervision

For many applications, norm regimentation is too restrictive, limiting the autonomy of agents. In such applications it is more desirable to allow agent to violate norms, but to compensate the violations with *sanctions*. In this section and



without loss of generality, we assume a single sanction event  $s$  to keep the presentation simple. We also assume that the unique sanction event does not appear in the behavior of the plant and that it is added to (imposed on) the plant by the supervisor after any norm-violating event. Finally, for  $w \in (\Sigma \cup \{s\})^*$  we use  $\hat{w}$  to denote  $w$  from which all occurrences of  $s$  are removed, e.g., if  $w = e_1e_2se_3se_4s$ , we have  $\hat{w} = e_1e_2e_3e_4$ . As before we assume a norm specification  $n \subseteq \Sigma^*$  and interpret its elements as sequences of events that cause a violation. We also assume the set of violating behaviors as extensions of norms, i.e.,  $Viol_n = n.\Sigma^*$ . Given a norm specification, we first define what it means to add sanctions after norm-violating event sequences in the norm specification.

**Definition 6. Sanctioned Norm.** *Let  $n \subseteq \Sigma^*$  be a norm,  $s$  be the sanction event, and  $w \in (\Sigma \cup \{s\})^*$ . The set of the sanctioned event sequences from  $n$ , denoted as  $San(n)$  and called sanctioned norm, is inductively defined as follows:*

- $\epsilon \in San(n)$ ; and
- if  $w \in San(n), w' \in \Sigma^*, \hat{w}.w' \in n$ , and for all  $w'' \sqsubset w'$  we have  $\hat{w}.w'' \notin n$ , then  $w.w'.s \in San(n)$ .

This definition ensures that a single sanction event is added after each violation. For example, if  $e_1, e_1e_2 \in n$ , then  $e_1s, e_1se_2s \in San(n)$ . Note that  $n = \{\hat{w} \mid w \in San(n) \setminus \{\epsilon\}\}$ . Next, we define the concept of *sanctioned behavior* which extends words in the sanctioned norm with events that do not cause any further norm violations.

**Definition 7. Sanctioned behavior.** *Let  $San(n)$  be a sanctioned norm. The set of sanctioned behavior, denoted as  $San^+(n)$ , is any non-violating extension of sanctioned norm. Formally  $San^+(n) = \{w.w' \mid w \in San(n), w' \in \Sigma^*, \forall w'' \sqsubset w' : \hat{w}.w'' \notin n\}$ .*

Given a norm specification, the set of sanctioned behaviors is included in the set of norm-compliant behaviors.

**Definition 8. Norm Compliant Behaviors.** *Let  $n \subseteq \Sigma^*$  be a norm and  $s$  be a sanction event. The set of  $n$ -compliant behaviors in presence of  $s$ , denoted as  $K_n^s = (\Sigma^* \setminus Viol_n) \cup San^+(n)$ , contains all non-violating and sanctioned behaviors.*

Note that omitting  $s$  from strings in  $San^+(n)$  results in the set of  $n$ -violating behaviors. Formally,  $Viol_n = \{\hat{w} : w \in San^+(n)\}$ . Observe that  $K_n^s$  consists of not only non-violating behaviors but also violating and sanctioned behaviors<sup>1</sup>. Moreover, as one can observe,  $K_n^s$  is defined *independently* of any specific plant.

In order to relate sanction-based compliant behaviors with a plant in which the sanction event does not occur, we first enrich the plant with the sanction event  $s$  and an auxiliary event  $\dot{s}$ , interpreted as no-sanction imposed. The introduction of

---

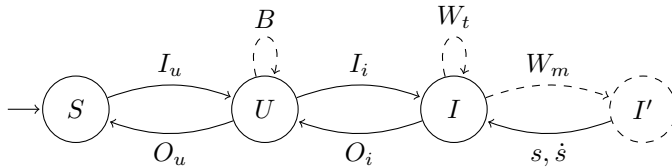
<sup>1</sup> We highlight that do not take sanctions to be permissible *events*, but consider an already sanctioned *behavior* as a norm-compliant one.

these two events enables us to prevent all violating behaviors (i.e., extensions of the norm without sanctions) while allowing norm compliant behaviors (including sanctioned behaviors). The natural question is how to enrich a plant with these two events. For this we, define the notion of sanction policy and virtual plant. A *sanction policy* determines when a sanction event may be added (i.e., after which sequences of events). A sanction policy  $F$  is therefore represented as a set of words, i.e.,  $F \subseteq (\Sigma \cup \{s, \dot{s}\})^*$ . Note that the words in  $F$  may contain different occurrences of  $s$  and  $\dot{s}$ . Also,  $F$  is defined independent of any norm or plant. Let  $\hat{w}$  be the same as  $w$  except that all occurrences of  $s$  and  $\dot{s}$  are removed. Note that  $\hat{w}$  is used before to remove all occurrences of  $s$ ; we have to extend this operation here to remove  $\dot{s}$  as well. We now define a virtual plant as a plant under specific sanction policy  $F$  by introducing states in the original plant where  $s$  and  $\dot{s}$  are the only available events.

**Definition 9. Virtual Plant.** A plant  $\mathcal{G}$  under the sanctioning policy  $F$ , denoted by  $F(\mathcal{G})$  and called virtual plant, generates the following behaviors:

$$L(F(\mathcal{G})) = \{w \in L(\mathcal{G}) \mid \forall w' \sqsubseteq w : w' \notin F\} \cup \{w.s.w', w.\dot{s}.w' \mid w, w' \in (\Sigma \cup \{s, \dot{s}\})^*, \hat{w}.w' \in L(\mathcal{G}), w \in F\}.$$

For our running example, one possible sanction policy is  $F = \{I_u I_i W_m\}$ . The following graph represents the virtual plant  $F(\mathcal{G})$  that extends the original plant  $\mathcal{G}$ .



The introduction of virtual plant enables us to reduce the notion of *sanctioning* into *regimentation* by assuming  $s$  and  $\dot{s}$  as the *only controllable events* in the plant under a given sanction policy. A supervisor can then control the plant under a given sanction policy by enabling either  $s$  or  $\dot{s}$ , but not both. As  $\dot{s}$  is an auxiliary event (interpreted as no-sanction is imposed), we can ignore this event in the plant’s behaviors. The set of behaviors of a virtual plant  $F(\mathcal{G})$  from which all occurrences of the auxiliary event  $\dot{s}$  are removed will be denoted as  $L_s^F(\mathcal{G})$ . It is crucial to note that this set includes all behaviors of the original plant as well as some new behaviors in which  $s$  occurs.

Given a plant  $\mathcal{G}$  and a sanctioning policy  $F$ , not all norms are sanctionable. This is mainly due to presence of uncontrollable events and their potential to result in violating behaviors. The following definition circumscribes the class of sanctionable norms under a specific sanctioning policy.

**Definition 10. Sanctionability.** A norm specified by  $n$  is sanctionable in  $\mathcal{G}$  under sanctioning policy  $F$ , if  $\overline{K}_n^s \cdot \Sigma_u \cap L_s^F(\mathcal{G}) \subseteq \overline{K}_n^s$ .

Note that we are considering a norm to be sanctionable under a sanction policy. Hence, a norm may be sanctionable in a plant under a sanction policy  $F$  but not under  $F' \neq F$ . In other words, we dismiss the problem of finding/constructing an appropriate sanction policy that guarantees the sanctionability of a norm in a plant. The following proposition highlights that if the sanctioning policy  $F$  imposes  $s$  or  $\dot{s}$  after any event, any norm will be sanctionable.

**Proposition 3. Dictatorial Sanctioning Policy.** *Any non-empty norm (specified by)  $n$  is sanctionable in plant  $\mathcal{G}$  under sanctioning policy  $F = \Sigma^*$ .*

*Proof.* In this case,  $F$  includes all possible words  $w \in \Sigma^*$ . In other words, (under  $F$ ) sanction operation  $s$  is imposed after any arbitrary event. Therefore, the sanctionability condition in Definition 10 always holds. Hence, for any nonempty  $n \subseteq \Sigma^*$ , we have that  $\overline{K}_n^s \cdot \Sigma_u \cap L_s^F(\mathcal{G}) \subseteq \overline{K}_n^s$ .

The next proposition states that no non-empty norm will be sanctionable if the sanctioning policy never imposes the sanction event.

**Proposition 4. Impotential Sanctioning Policy.** *Any norm specified by a non-empty  $n$  is sanctionable in plant  $\mathcal{G}$  under sanctioning policy  $F = \emptyset$ .*

*Proof.* In this case,  $F$  includes no word  $w \in \Sigma^*$ . In other words, sanction operation  $s$  can be imposed after no event. Therefore, the sanctionability condition in Definition 10 never holds. I.e. for no non-empty  $n \subseteq \Sigma^*$ , we have that  $\overline{K}_n^s \cdot \Sigma_u \cap L_s^F(\mathcal{G}) \subseteq \overline{K}_n^s$ .

We now define the notion of sanction-based supervisor as a means of suppressing norm violating behaviors in a multi-agent system.

**Definition 11. Sanction-Based Norm Supervisor.** *A sanction-based norm supervisor for a multi-agent system  $\mathcal{G}$  under a sanctioning policy  $F$ , is a function of the form  $V_s : L(F(\mathcal{G})) \mapsto \{ \Sigma, \{s\}, \{\dot{s}\} \}$ , where  $V_s(w)$  denotes the set of events that are enabled next.*

Note that the set of events that a sanction-based norm supervisor enables includes all plant events (interpreted as non-controllable events) with either the controllable event  $s$  or the controllable event  $\dot{s}$ . This type of supervisor allows violating behaviors to take place but may impose the sanction event in order to punish violations.

**Definition 12. Sanction-Based Supervisor.** *Let  $\mathcal{G}$  be a multi-agent system,  $F$  a sanctioning policy, and  $V_s$  a sanction-based norm supervisor for  $\mathcal{G}$  under  $F$ . The sanctioned language of  $\mathcal{G}$  under  $V_r$  is defined as  $L(V_s/\mathcal{G}) = \{w.\sigma \mid w.\sigma \in L(F(\mathcal{G})), w \in L(V_s/\mathcal{G}), \sigma \in V_s(w)\} \cup \{\epsilon\}$ .*

We emphasize that while our formerly introduced regiment-based norm supervisor uses the “real” behavior of the multi-agent system, the sanction-based norm supervisor considers a specific multi-agent behavior that is “virtually” extended under a given sanctioning policy. This is mainly because after

(virtually) extending the multi-agent system under a policy, potential behaviors may include sanction events while they are not (really) events generated by the plant.

**Theorem 2.** *Let  $\mathcal{G}$  be a multi-agent system,  $F$  a sanctioning policy, and  $n \subseteq \Sigma^*$  be a norm such that  $K_n^s \neq \emptyset$ . Then, there exists a sanction-based norm supervisor  $V_s$  such that  $L(V_s/\mathcal{G}) = \overline{K_n^s}$  iff norm  $n$  is sanctionable in  $\mathcal{G}$  under  $F$ .*

*Proof.* The line of proof is analogous to the proof of Theorem 1 if we just replace  $L(\mathcal{G})$  by the virtually extended behavior  $L(F(\mathcal{G}))$  under sanctioning policy  $F$ .

In this sanction-based interpretation of supervision,  $n$  is enforceable when  $n$ -violating behavior can be sanctioned. Note that enforcing a norm via sanctions allows “bad” behaviors (of the multi-agent system) to take place after which sanction will incur. In our running example, any behavior that ends with  $W_m$  is a norm violating behavior. Using a sanctioning policy  $F = \{\Sigma^*.W_m\}$ , this theorem shows the existence of a sanction-based supervisor that imposes sanctions after any occurrence of  $W_m$ . It is observable that such a sanctioning mechanism does not prevent any norm-compliant behavior, e.g., watching a tutorial is now possible for the student (without paying any sanction). Given a plant, a norm may be not sanctionable under a specific sanction policy. Here, we define the concept of *supremal sanctionability* as the largest set of sanctionable behaviors (under a given sanction policy) in a plant.

**Definition 13. Supremal Sanctionability.** *Let  $\mathcal{G}$  be a multi-agent system,  $n \subseteq \Sigma^*$  a norm such that  $K_n^s \neq \emptyset$ , and  $F$  a sanctioning policy. The supremally sanctionable  $n$ -compliant behavior in  $\mathcal{G}$  under  $F$ , denoted  $K_n^{F,s\uparrow}$ , is defined as  $K_n^{F,s\uparrow} = \bigcup_{K \in R(K_n^s)} K$ , where  $R(K_n^s) = \{K \subseteq K_n^s : \overline{K} \cdot \Sigma_u \cap L_s^F(\mathcal{G}) \subseteq \overline{K}\}$ .*

That is,  $K_n^{F,s\uparrow}$  represents the largest sublanguage of  $K_n^s$  that satisfies the sanctionability condition (Definition 10). Next, we point out that, at the technical level, we have basically transformed a norm enforcement via sanctions problem into a norm regimentation task, albeit in a modified multi-agent system plant. Doing so allows us to directly import Proposition 2 to the sanction-based framework.

**Proposition 5.** *Let  $\mathcal{G}$  be a multi-agent system,  $F$  a sanction policy for which there exists a generator  $\mathcal{G}_F$  such that  $L(\mathcal{G}_F) = F$ , and  $n \subseteq \Sigma^*$  a norm for which there exists a generator  $\mathcal{G}_n$  such that  $L(\mathcal{G}_n) = n$ . Suppose further that  $K_n^s \neq \emptyset$ . Then, a generator  $\mathcal{R}$  such that  $L(\mathcal{R}) = K_n^{F,s\uparrow}$  can be computed in polynomial time w.r.t.  $\mathcal{G}$ ,  $\mathcal{G}_F$  and  $\mathcal{G}_n$ .*

*Proof.* Because norm specification  $n$  is regular and implemented with  $\mathcal{G}_n$ , sets  $Viol_n^s$  and  $K_n^s$  are regular as well. Since  $F$  is regular and generated by  $\mathcal{G}_F$ , we can once again directly apply the results in [26] and the thesis follows.

## 6 Reparation-Based Supervision

In this section, we consider some system events as “repairing” events, in the sense that any violating behavior followed by such events is considered as non-violating behavior. In our running example, one may consider watching a tutorial as repairing the violation of watching a movie. In addition to our assumption that the set of events consists of disjoint sets of controllable  $\Sigma_c$  and uncontrollable events  $\Sigma_u$ , we introduce an orthogonal partitioning over the set of events which makes distinction between reparation events  $\Sigma_p$  and non-reparation events  $\Sigma_{np}$ , i.e.,  $\Sigma = \Sigma_p \cup \Sigma_{np}$ . For simplicity and without loss of generality, we assume  $\Sigma_p = \{p\}$ . Also, we only consider *immediate* reparation; we do not allow distant reparation. Allowing multiple (non-reparation) events between a violating behavior and its repair calls for a method that clarifies how to deal with multiple violations and their reparation priorities. Moreover, the specification of norms will be constrained to use only non-reparation events from  $\Sigma_{np}$ .

**Definition 14. Norm Violating and Compliant behaviors.** *Let  $n \subseteq \Sigma_{np}^*$  be a norm and  $p$  be a repair event. The set of  $n$ -violating behaviors in presence of  $p$  is  $Viol_n^p = n \cdot \Sigma_{np} \cdot \Sigma^*$ , and the set of  $n$ -compliant behavior in presence of  $p$  is  $K_n^p = \Sigma^* \setminus Viol_n^p$ .*

Note that this view internalizes the reparation of norm violating behavior. Hence, we do not need any external set of sanction operations (as it was the case in Definition 8).

**Definition 15. Repairability.** *A norm specified by  $n$  is repairable in  $\mathcal{G}$  with event  $p$  if  $\overline{K}_n^p \cdot \Sigma_u \cap L(\mathcal{G}) \subseteq \overline{K}_n^p$ .*

In our running university example, norm  $n = \Sigma^* \cdot W_m$  is not repairable. This is because after some norm-compliant behaviors in  $\overline{K}_n^p$ , the occurrence of uncontrollable event  $W_m$  results in norm-violating behaviors (that are obviously not in  $\overline{K}_n^p$ ). We later present a brief abstract example in which the norm is repairable.

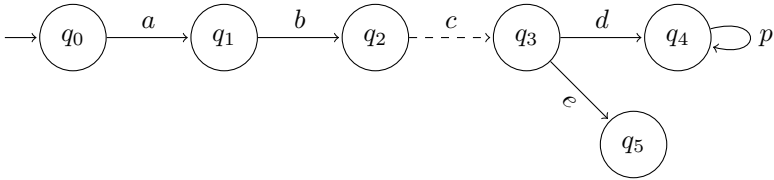
We now define the notion of reparation-based supervisor as a mean of enabling the multi-agent system to repair possible violating behaviors.

**Definition 16. Reparation-Based Norm Supervisor.** *A reparation-based supervisor for a multi-agent system  $\mathcal{G}$  is a function of the form  $V_p : L(\mathcal{G}) \mapsto \{\Sigma_e \mid \Sigma_e \in 2^\Sigma, \Sigma_u \subseteq \Sigma_e\}$ , where  $V_p(w)$  denotes the set of events that are enabled (i.e., allowed) next.*

This type of supervisor can now allow violating behaviors that are immediately followed by a repair event, while regimenting all other violating behaviors.

**Definition 17. Reparation-Based Supervision.** *Let  $\mathcal{G}$  be a multi-agent system,  $p$  be the repair event, and  $V_p$  a reparation-based norm supervisor for  $\mathcal{G}$ . The supervision of  $\mathcal{G}$  by  $V_p$  obtains a multi-agent system, denoted as  $V_p/\mathcal{G}$ , that generates behaviors specified as  $L(V_p/\mathcal{G}) = \{\epsilon\} \cup \{w \cdot \sigma \cdot p \in L(\mathcal{G}) \mid w \in L(V_p/\mathcal{G}), w \cdot \sigma \in n\} \cup \{w \cdot \sigma \in L(\mathcal{G}) \mid w \in L(V_p/\mathcal{G}), w \cdot \sigma \notin n\}$ .*

In reparation-based norm supervision,  $n$  is enforceable when violating behaviors that can not be immediately repaired are avoided. I.e., enforcing a norm via this approach only sees a behavior  $n$ -violating if after the occurrence of the norm, it is not immediately repaired. As shown in Sect. 4, in order to avoid violating behaviors, the regiment-based supervision of a plant may result in a subset of norm-compliant behavior by disallowing all the events that may result in violating behavior. In comparison, the reparation-based supervisor only disallows the events after which there is no reparation event in the plant. For instance, for  $\Sigma = \{a, b, c, d, e, p\}$ ,  $\Sigma_u = \{c\}$ ,  $\Sigma_p = \{p\}$ , and  $n = \{abcd\}$ , applying the regiment-based vision after a sequence of events  $w = a$  results in losing  $n$ -compliant behavior  $abce$  in the following plant while the reparation-based approach allows it to take place. Although allowing  $b$  may result in  $n$ -violating behavior  $abcd$ , reparation is available afterwards.



**Theorem 3.** *Let  $\mathcal{G}$  be a multi-agent system,  $p$  be the repair event, and  $n \subseteq \Sigma^*$  be a norm such that  $K_n^p \neq \emptyset$ . There exists a reparation-based norm supervisor  $V_p$  such that  $L(V_p/\mathcal{G}) = \overline{K_n^p}$  iff norm  $n$  is repairable in  $\mathcal{G}$ .*

*Proof.* The line of proof is analogous to the proof of Theorem 1 if we just replace  $L(\mathcal{G})$  by  $L(V_p/\mathcal{G})$ .

In our running example, any behavior that ends with  $W_m$  is a norm violating behavior. As the reparation-based supervisor is designed to allow norm-violating behaviors to take place, it does not prevent the event of watching a movie if this event is immediately followed by the repair event, which is in this case watching a tutorial  $W_t$ . The above theorem suggests the existence of a repair-based supervisor that ensures violations are either not take place or they are immediately followed by a repair event. It is observable that a repair-based supervisor is similar with a regimentation-based supervisor with a one-step look ahead function.

In a specific plant, a norm may be not repairable if it does not pass the repairability condition in Definition 15. Here, we define the concept of *supremal repairability* as the largest set of repairable behaviors in a plant.

**Definition 18. Supremal Repairability.** *Let  $\mathcal{G}$  be a multi-agent system,  $n \subseteq \Sigma^*$  a norm such that  $K_n^p \neq \emptyset$ , and  $p$  the repairing event. The supremally repairable  $n$ -compliant behavior in  $\mathcal{G}$  with  $p$ , denoted  $K_n^{p\uparrow}$ , is defined as  $K_n^{p\uparrow} = \bigcup_{K \in R(K_n^p)} K$  where  $R(K_n^p) = \{K \subseteq K_n^p : \overline{K} \cdot \Sigma_u \cap L(\mathcal{G}) \subseteq \overline{K}\}$ .*

That is,  $K_n^{p\uparrow}$  represents the largest sublanguage of  $K_n^p$  that satisfies the repairability condition (Definition 15). The following result mirrors that in Propositions 2 and 5 for the case of repairability.

**Proposition 6.** *Let  $\mathcal{G}$  be a multi-agent system,  $p$  a repair event, and  $n \subseteq \Sigma^*$  a norm for which there exists a generator  $\mathcal{G}_n$  such that  $L(\mathcal{G}_n) = n$ . Suppose further that  $K_n^p \neq \emptyset$ . Then, a generator  $\mathcal{R}$  such that  $L(\mathcal{R}) = K_n^{p\uparrow}$  can be computed in polynomial time w.r.t.  $\mathcal{G}$  and  $\mathcal{G}_n$ .*

## 7 Related Work

Our proposal contributes to the literature on normative multi-agent systems by re-purposing models and results from SCT to provide a formal, implementable and tractable semantics for the key normative concepts, such as norms and norm enforcement mechanisms. It also contributes to the literature of SCT as it provides a novel application domain, and the notions of sanctioning and repairing that may be applicable to control processes. In the normative multi-agent systems literature, various approaches have been proposed to model norms and norm enforcement mechanisms. Some approaches focus on logical characterization of normative multi-agents systems while others aim at designing frameworks to develop normative multi-agent systems. For example, [1] uses a linear-time temporal logic to represent norms and system behaviors. In such work, the idea of norm enforcement, which is enriched with lookahead possibilities, is characterized as decision problems with respect to specific classes of norms. Similar to our regiment-based supervisor function, [1] uses a guard function, that enables/disables options that (could) violate norms after a system history. However, the authors consider norm monitors to be imperfect while we assume that our supervisors have perfect observability over the behavior of the MAS. They also do not consider sanction- and repair-based norm enforcement mechanisms.

The idea of sanction-based enforcement mechanisms has been studied in several works, e.g., [3, 12, 14]. There, multi-agent systems are semantically modeled as transitions systems, where traces are interpreted as system behaviors. The enforcement of norms by means of sanctions is realized by identifying and sanctioning violating behaviors. This is done by modifying the valuation of states that occur in the violating behaviors. Our sanction-based approach is closely related except that in our approach sanctioning events are added to behaviors which are defined as event sequences.

Other approaches concern the development of normative multi-agent systems. For example, [16, 18] takes into account the organizational structure of multi-agent systems in order to develop middle-wares for normative multi-agent organizations/institutions while we dismiss agent hierarchies and see a MAS as a plant that generates strings of discrete events. Moreover, [4] makes a distinction between regulative (deontic) and substantive (constitutive) norms while we define norms to be any arbitrary sub-language of event strings. Finally, [2] builds on the idea of norm enforcement and proposes a programming approach

to develop norm-aware agents. These agents can deliberate on enforceable norms to decide whether they follow or violate the norms.

Our work clearly draws from (and hence is closely related to) some approaches in the DESs, e.g., [13], and SCT, e.g., [21,22]. We build on the *controllability condition* in [13] to introduce our three types of norm enforcement, namely, regimentation-, sanction-, and repair-based enforcement. However, we extend the classic concept of supervisor in DESs as a *violation precluding mechanism* by considering supervisors that allow violations to take place but are also able to impose sanction operations. This approach, i.e., allowing violations to occur, leads to a toolbox of norm enforcement mechanisms that are applicable in contexts that call for higher level of agent autonomy. Moreover, we introduce a normative dimension into reasoning about behavior of DESs. One noticeable contribution that considers decentralized control in multi-agent systems using SCT is [6] which mainly focuses on reformulating the results of SCT in terms of model checking problems in an epistemic temporal logic.

We would like to emphasize that our work differs from related but distinguishable approaches proposed in [7,28]. In our approach, we focus on coordination of a multi-agent systems, using the concept of norm-supervisors, in order to avoid/suppress some undesired but system-independent norm-violating behaviors. In contrast, [7,28] shows that some desired properties such as non-blocking (in a class of resource allocation systems) can be achieved using a supervisor that controls the plant's behavior. Although this approach might be similar to our regiment-based norm supervising mechanism, it should be emphasized that our norms are plant-independent. Hence, they do not necessarily reflect properties that are "good" in a plant, but represent "good" behaviors regardless of any plant.

## 8 Conclusion and Future Work

This paper presents a formal framework rooted in Supervisory Control Theory (SCT) and normative multi-agent systems. We show that three well-known types of norm enforcement mechanisms, namely, regimentation-, sanction-, and repair-based enforcement, can be modeled as special supervisor from SCT. Importing the controllability theorem from SCT, we prove the existence of supervisors that can either prevent, sanction or repair violating behaviors in a given multi-agent system. In addition to providing a semantics for norm and norm enforcement in formal languages, this work supports the direct use of available tools for SCT, such as TCT/STCT [27] or SUPREMICA [20] to synthesize supervisors. We intend to run experiments by developing norm-based supervisory systems using these tools. The fact is that by restricting to generator-based "regular" systems and properties, which are still fairly expressive, implementing SCT-based normative systems becomes amenable for computation (e.g., can be realized via the existing tools mentioned above) [9]. Hence, the possibility of automatically synthesizing SCT-based norm enforcement mechanisms that can be used for on-line norm monitoring/enforcement in multi-agent systems becomes a feasible.



Regarding the practicality of our approach we emphasize that, since we base our norm specification on given sequences of events, our presented norm enforcement mechanisms are applicable in domains where traces of violating behaviors are accessible (e.g., using data-oriented behavior models). We believe that such a set can be collected in big data projects, for example, by using data mining to extract and categorize instances of event sequences that are (un)desirable in a specific multi-agent system.

As future research, one can relax the full observability assumption and study the enforcement of norms under partial observability. In this line, dynamics of supervisor's belief-level can be incorporated by taking into account multiple belief worlds that are linked using *epistemic* event. An alternative approach would be to use a network of supervisors with local observability. Then having a class of communication events can enable them to collectively supervise the plant. Another possible extension is to relax the restriction on *immediate* reparations in reparation-based supervising mechanisms.

In this work, we merely focused on individual agents and reasoned about the normative behavior of a multi-agent plant in which supervisors can bring about desired behaviors by means of regimentation, sanctioning, or reparation mechanisms that controls individuals. The transition to collective actions, e.g. using concurrent structures, is left for future work. Such an extension may result in the introduction of mechanisms that take into account group potentials (e.g., for making collusion) and possible semantics of normative concepts in relation to collective actions.

Another possible extension would be to formalize “hybrid” supervisors. For instance, a supervisor that sanctions violating behaviors for a certain number of times, e.g., just once, and then starts preventing any further violations. In this case we are sensitive to sequence (and number) of imposed sanction operations. Roughly speaking, when the plant generates violating behaviors for which we already sanctioned it before, its behavior may trigger a meta-norm that allows the supervisor to use regiment-based supervision. Such a mixed supervisor may find application in some domains that can tolerate norm-violating behavior only to a given threshold, e.g., in safety-critical computer/industrial systems [11,25].

## References

1. Alechina, N., Bulling, N., Dastani, M., Logan, B.: Practical run-time norm enforcement with bounded lookahead. In: Proceeding of the 2015 International Conference on Autonomous Agents and Multiagent Systems, pp. 443–451 (2015)
2. Alechina, N., Dastani, M., Logan, B.: Programming norm-aware agents. In: Proceeding of the 11th International Conference on Autonomous Agents and Multiagent Systems, vol. 2, pp. 1057–1064 (2012)
3. Alechina, N., Dastani, M., Logan, B.: Reasoning about normative update. In: Proceeding of the Twenty-Third International Joint Conference on Artificial Intelligence, pp. 20–26 (2013)

4. Alvarez-Napagao, S., Aldewereld, H., Vázquez-Salceda, J., Dignum, F.: Normative monitoring: semantics and implementation. In: De Vos, M., Fornara, N., Pitt, J.V., Vouros, G. (eds.) COIN -2010. LNCS, vol. 6541, pp. 321–336. Springer, Heidelberg (2011). doi:[10.1007/978-3-642-21268-0\\_18](https://doi.org/10.1007/978-3-642-21268-0_18)
5. Andrighetto, G., Governatori, G., Noriega, P., van der Torre, L.W.: Normative Multi-Agent Systems, vol. 4. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Germany (2013)
6. Aucher, G.: Supervisory control theory in epistemic temporal logic. In: International conference on Autonomous Agents and Multi-Agent Systems, pp. 333–340 (2014)
7. Barkaoui, K., Chaoui, A., Zouari, B.: Supervisory control of discrete event systems based on structure theory of petri nets. In: 1997 IEEE International Conference on Systems, Man, and Cybernetics, 1997 Computational Cybernetics and Simulation, vol. 4, pp. 3750–3755. IEEE (1997)
8. Bicchieri, C.: The Grammar of Society: The Nature and Dynamics of Social Norms. Cambridge University Press, New York (2005)
9. Blondel, V.D., Tsitsiklis, J.N.: A survey of computational complexity results in systems and control. *Automatica* **36**(9), 1249–1274 (2000)
10. Boella, G., van der Torre, L.W.: Regulative and constitutive norms in normative multiagent systems. *KR* **4**, 255–265 (2004)
11. Bowen, J., Stavridou, V.: Safety-critical systems, formal methods and standards. *Softw. Eng. J.* **8**(4), 189–209 (1993)
12. Bulling, N., Dastani, M.: Norm-based mechanism design. *Artif. Intell.* **239**, 97–142 (2016)
13. Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems. Springer Science & Business Media, US (2009)
14. Dastani, M., Grossi, D., Meyer, J.C.: A logic for normative multi-agent programs. *J. Log. Comput.* **23**(2), 335–354 (2013)
15. Dastani, M., Grossi, D., Meyer, J.-J.C., Tinnemeier, N.: Normative multi-agent programs and their logics. In: Meyer, J.-J.C., Broersen, J. (eds.) KRAMAS 2008. LNCS, vol. 5605, pp. 16–31. Springer, Heidelberg (2009). doi:[10.1007/978-3-642-05301-6\\_2](https://doi.org/10.1007/978-3-642-05301-6_2)
16. Esteva, M., Rosell, B., Rodríguez-Aguilar, J.A., Arcos, J.L.: AMELI: an agent-based middleware for electronic institutions. In: 3rd International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 236–243 (2004)
17. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory Languages and Computation. Addison-Wesley, Reading (1979)
18. Hübner, J.F., Sichman, J.S., Boissier, O.:  $\mathcal{S} - Moise^+$ : A middleware for developing organised multi-agent systems. In: Boissier, O., Padget, J., Dignum, V., Lindemann, G., Matson, E., Ossowski, S., Sichman, J.S., Vázquez-Salceda, J. (eds.) AAMAS 2005. LNCS, vol. 3913, pp. 64–77. Springer, Heidelberg (2006). doi:[10.1007/11775331\\_5](https://doi.org/10.1007/11775331_5)
19. y López, F.L., Luck, M.: Modelling norms for autonomous agents. In: 4th Mexican International Conference on Computer Science (ENC 2003), 8–12 September 2003, Apizaco, Mexico, pp. 238–245 (2003)
20. Åkesson, K., Fabian, M., Flordal, H., Vahidi, A.: Supremica - a tool for verification and synthesis of discrete event supervisors. In: Proceeding of the 11th Mediterranean Conference on Control and Automation (2003)
21. Ramadge, P.J., Wonham, W.M.: Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.* **25**(1), 206–230 (1987)

22. Ramadge, P.J., Wonham, W.M.: The control of discrete event systems. *Proc. IEEE* **77**(1), 81–98 (1989)
23. Reiser, C., da Cunha, A., Cury, J.: The environment GRAIL for supervisory control of discrete event systems. In: *Proceeding of 8th International Workshop on Discrete Event Systems*, pp. 390–391, July 2006
24. Ricker, L., Lafortune, S., Gene, S.: DESUMA: A tool integrating GIDDES and UMDDES. In: *Proceeding of 8th International Workshop on Discrete Event Systems*, pp. 392–393 (2006)
25. Storey, N.R.: *Safety Critical Computer Systems*. Addison-Wesley Longman Publishing Co Inc., Reading (1996)
26. Wonham, W.M., Ramadge, P.J.: On the supremal controllable sublanguage of a given language. *SIAM J. Control Optim.* **25**(3), 637–659 (1987)
27. Zhang, Z., Wonham, W.M.: STCT: An efficient algorithm for supervisory control design. In: *Symposium on Supervisory Control of Discrete Event Systems*, pp. 249–6399 (2001)
28. Zouari, B., Barkaoui, K.: Parameterized supervisor synthesis for a modular class of discrete event systems. In: *Proceeding of the IEEE International Conference on Systems, Man & Cybernetics*, 5–8 October 2003, Washington, D.C, USA, pp. 1874–1879 (2003)