ECAI 2016
G.A. Kaminka et al. (Eds.)

doi:10.3233/978-1-61499-672-9-751

Distributed Controllers for Norm Enforcement

Bas Testerink and **Mehdi Dastani**[1] and **Nils Bulling**[2]

Abstract. This paper focuses on computational mechanisms that control the behavior of autonomous systems at runtime without necessarily restricting their autonomy. We build on existing approaches from runtime verification, control automata, and norm-based systems, and define norm-based controllers that enforce norms by modifying system behavior at runtime to make it norm compliant. For many applications, an autonomous system should comply with a set of norms. We extend our approach to a distributed setting, where a set of norm-based controllers jointly modify the runtime behavior of an autonomous system. The norms that a set of norm-based controllers jointly enforce are investigated and characterized in terms of the norms that are enforced by individual norm-based controllers. We show that a set of norm-based controllers is able to modify the runtime behavior of an autonomous system to make it compliant with all norms that the individual norm-based controllers aim at enforcing.

1 Introduction

The emergence of autonomous systems requires mechanisms to monitor and control their behavior in order to ensure system level properties. Norms are a popular candidate for the specification of system level properties and can be seen as standards of behavior that distinguish good and bad behavior (see e.g., [25, 14, 6]). Various languages have been proposed to represent different classes of norms such as state-based norms, action-based norms, temporal norms, and for each class of norms, monitoring and enforcement models have been proposed to detect and control norm violations, respectively (e.g., [8, 1, 11, 5]).

Existing proposals for norm monitoring and enforcement are either based on logical models where norm violations are explained in terms of the satisfiability of a violation formula (i.e. a formula that characterizes violated states or executions) and norm enforcement is explained in terms of model updates [1, 3], or are concerned with practical frameworks for building norm-based systems (e.g., [16, 13, 7]). Most logical approaches to norm enforcement focus on infinite executions and are not concerned with runtime norm enforcement. An exception is the runtime model for norm enforcement proposed in [2]. However, in this work norms are only enforced by halting the system execution before a norm violation occurs. This paper differentiates from earlier work, both by ourselves and others, by focusing on a theoretical analysis of (distributed) runtime norm enforcement rather than an offline analysis on the effects of norm compliance of a system.

In other branches of computer science, such as runtime verification and control automata, the idea of monitoring and control at runtime has been extensively studied, albeit from a different perspective. The main research problem in these areas is to ensure some system level properties for an untrusted system, also called the target system or the plant, when the system is expected to produce unwanted behavior. In these research areas runtime controllers are introduced to revise the behavior of the untrusted system to ensure the given system level properties. Such a controller reviews the actions that the system produces and may decide per action whether to allow the action, suppress the action or execute extra actions. These controllers have formally been studied, e.g., in [24, 21, 17, 9].

A characteristic feature of norms is that they may be enforced by means of regimentation and sanctioning. A norm can be either regimented in the sense that norm violations are prevented, or sanctioned in the sense that norm violations incur sanctions. Norm enforcement by means of sanctioning respects the autonomy of the controlled system as it allows violations to take place, but may intervene by adding corrective/repair actions. This paper builds on runtime controllers and applies them to norm enforcement. Regimenting and sanctioning controllers are distinguished, depending on whether norm regimentation or norm sanctioning is applied. We will use the term norm-based controller to indicate a controller that is enforcing a norm through regimentation or sanctioning. Our first contribution is to model norm regimentation and sanctioning in a consistent manner with respect to the aforementioned controller models of [21] or [17]. These models allow us to formally analyze regimenting and sanctioning controllers, and investigate the runtime enforcement of norms. A norm-based controller is defined as a computational entity that reviews the actions performed by the target system. If there is a violation about to happen, then the system execution is halted in case of regimentation, or the violating action is allowed and followed by a corrective/repair action in case of sanctioning.

For many applications, a target system should be compliant with a set of norms. This has led to various proposals for distributed architectures for norm enforcement (cf.[22, 18, 23, 28]). The general setup in these architectures is that multiple norm-based control mechanisms are applied concurrently on a target system. The benefits of concurrently applied control mechanisms include the removal of a single point of failure and a possible bottleneck at some central control mechanism. Concurrently applied control mechanisms independently process local observations and communicate them in order to collaborate on the task of enforcing norms. We follow this general architectural setup and use the term distributed norm-based controller to refer to a set of norm-based controllers. Our second contribution is the introduction of a framework that allows us to formally analyze a set of norm-based controllers that enforce their norms concurrently on one and the same target system. We propose the construction of collaborative automata to show that a set of norm-based controllers is able to modify the behavior of a target system to comply with the entire set of norms.

Section 2 discusses the relevant background theory. Section 3 pro-

[1] Utrecht University, The Netherlands, email: {B.J.G.Testerink,M.M.Dastani}@uu.nl
[2] TU Delft, The Netherlands, email: N.Bulling@tudelft.nl

poses our model for norms and norm enforcement. In Section 4 controller automata are introduced to specify norm-based controllers. In Section 5 collaborative automata are defined and analyzed. Finally Section 6 discusses related approaches and provides concluding remarks.

2 Property Enforcement

Throughout this paper we model the possible actions of a target system by a fixed global, non-empty set of actions $\mathcal{A}$. We denote with $\mathcal{A}^*$ the finite sequences of elements of $\mathcal{A}$, referred to as *words*, including the empty word ε. For two words $\alpha, \alpha' \in \mathcal{A}^*$ we denote with $\alpha' \preceq \alpha$ and $\alpha' \prec \alpha$ that α' is a prefix and strict prefix of α, respectively. We assume a given equivalence relation $\sim \subseteq \mathcal{A}^* \times \mathcal{A}^*$, called *similarity relation*, that indicates when two words are semantically equivalent. We use $\alpha_{..i}$ and $\alpha_{j..}$ as the prefix of α up to and including index i and the suffix of α from and including index j, respectively. For two words $\alpha, \alpha' \in \mathcal{A}^*$ we use $\alpha\alpha'$ for the concatenation of α and α'. The next example illustrates the concepts.

Example 1 (Scenario) *Consider a file system where an agent can manipulate a file. The possible actions are* r *(read),* w *(write),* s *(save) and* b *(backup), i.e.* $\mathcal{A} = \{r, w, s, b\}$. *The agent can attempt to execute these actions in any order and as often as it desires. It might be prescribed by the system designer that the file must be saved in-between any two write actions. That would mean that* $rwrsw$ *is a good word but* $rwrws$ *is a bad word. Moreover, the words* s *and* ss *might be considered semantically equivalent, thus* $s \sim ss$.

We have opted for a simple scenario for clarity and conciseness of the examples. However, we emphasize that the contributions of this paper are applicable to more complex scenarios where concurrent processes perform sequences of actions that might be considered good or bad. Other example scenarios might be for instance electronic markets, social media and conference management systems.

2.1 Properties

We follow [21] and define a property as a set of words that are assumed to represent allowed system behaviors. We impose two constraints on a property. It must contain the empty word and must be closed under the similarity relation $\sim$.

Definition 1 (Property, P**)** *A property is given by* $\mathsf{P} \subseteq \mathcal{A}^*$ *such that* $\varepsilon \in \mathsf{P}$ *and if* $\alpha \in \mathsf{P}$ *and* $\alpha \sim \alpha'$ *then also* $\alpha' \in \mathsf{P}$ *for all* $\alpha, \alpha' \in \mathcal{A}^*$.

Example 2 (Ex. 1 cont., Property) *Let property* P *contain all words where in-between every two write actions* w *there is a save action* s. *Therefore the word* $rwrws$ *is not in* P, *whilst* $rwrsws$ *and* rwr *are in* P.

2.2 Controllers and Enforcement

A controller reviews an input word from $\mathcal{A}^*$ of the target system from left to right and produces some output word from $\mathcal{B}^*$ defined over some action set $\mathcal{B}$. It can be the case that $\mathcal{A}^* = \mathcal{B}^*$, but also that the controller introduces new actions that cannot be produced by the target system. A controller can deterministically revise the word into a new word. Thus, a controller can be seen as a mechanism to revise action executions at run-time. As such, a decision to previously output a word cannot be reverted in the future. This is captured by the formal definition below.

Definition 2 (Controller, m**)** *A* controller (function) *over* $(\mathcal{A}, \mathcal{B})$ *is given by a function* $\mathsf{m} : \mathcal{A}^* \to \mathcal{B}^*$ *such that if* $\alpha' \preceq \alpha$ *then* $\mathsf{m}(\alpha') \preceq \mathsf{m}(\alpha)$ *for all* $\alpha, \alpha' \in \mathcal{A}^*$.

In the following, we assume, if not said otherwise, that a property P over $\mathcal{A}^* \cup \mathcal{B}^*$, a controller m over $(\mathcal{A}, \mathcal{B})$, and a word $\alpha \in \mathcal{A}^*$ are given. We simply say a controller over $\mathcal{A}$ whenever $\mathcal{A} = \mathcal{B}$.

Example 3 (Ex. 2 cont., Controller) *Let* m *be a controller over* $\mathcal{A}$. *Consider* $\alpha = rwr$ *and assume* $\mathsf{m}(\alpha) = \alpha' = rwsr$. *This implies that given* $\alpha'' = rwrs$ *it cannot be the case that* $\mathsf{m}(\alpha'') = \alpha''$, *because* $\alpha \preceq \alpha''$ *but* $\mathsf{m}(\alpha) \not\preceq \mathsf{m}(\alpha'')$ *(i.e.* $rwsr \not\preceq rwrs$*). Intuitively, when* m *is reviewing the word* α'' *and reaches the second read action, then it inserts a save action, because* $\mathsf{m}(rwr) = rwsr$. *Then, when it reviews the* s *action in* α'' *it cannot retract the inserted save action.* $rwsr$ *has to be a prefix of the word that* m *returns given* α''.

The intended purpose of a controller is that it revises all words from $\mathcal{A}^*$ such that the revisions satisfy some desired property P (i.e. $\mathsf{m}(\alpha) \in \mathsf{P}$ for all $\alpha \in \mathcal{A}^*$). This is called soundness [21]. A trivial way to achieve *soundness* is to revise any word to the empty word. However, we want a controller to not only ensure correct behavior, but also to be *transparent* in the sense that the target system's behavior is not meaningfully altered if there is no violation of the property [17]. In other words, if a word already satisfies the property, then it should remain unchanged or at least be revised to a similar word wrt. $\sim$. Otherwise the word is mapped to its longest prefix satisfying the property.

Definition 3 (Longest Correct Prefix) *The longest correct prefix of* $\alpha \in \mathcal{A}^* \cup \mathcal{B}^*$ *wrt.* P *is* $\alpha' \in \mathsf{P}$ *such that* $\alpha' \preceq \alpha$ *and for all* $\alpha'' \in \mathcal{A}^* \cup \mathcal{B}^*$ *with* $\alpha' \prec \alpha'' \preceq \alpha$ *it holds that* $\alpha'' \notin \mathsf{P}$.

Next we recall the definition of precise enforcement from [21]. If a controller precisely enforces a property, then any action in any correct input word must (immediately) be allowed.

Definition 4 (Precise Enforcement, [21]) m *precisely enforces* P *if and only if for all* $\alpha \in \mathcal{A}^*$ *the following holds:*

1. $\mathsf{m}(\alpha) \in \mathsf{P}$ *and*
2. *if* $\alpha \in \mathsf{P}$ *then for all* $\alpha' \in \mathcal{A}^*$ *with* $\alpha' \preceq \alpha$ *it holds that* $\mathsf{m}(\alpha') = \alpha'$.

Example 4 (Ex. 1 cont., Precise Enforcement) *Let* P *be the property that contains all words in which each occurrence of* w *is preceded by* b *(before each write operation a backup must be performed). Let* m *be the controller over* $\mathcal{A}$ *that revises word* $\alpha \in \mathcal{A}^*$ *as follows: if there exists a minimal index* i *such that* $\alpha[i] \neq b$ *and* $\alpha[i+1] = w$ *then* $\mathsf{m}(\alpha) = \alpha_{..i}$; *otherwise* $\mathsf{m}(\alpha) = \alpha$. *That is, if the agent tries to write without performing a backup, then* m *will suppress the write action as well as all subsequent actions.* m *precisely enforces* P.

It is well known that exactly the class of safety properties from the safety-progress classification of [10] can be precisely enforced [21], where property P over $\mathcal{A}^*$ is a *safety property* if for all $\alpha \in \mathcal{A}^*$ with $\alpha \notin \mathsf{P}$ it holds that $\alpha \cdot \alpha' \notin \mathsf{P}$ for all $\alpha' \in \mathcal{A}^*$ [20]. We observe that if a controller m over $(\mathcal{A}, \mathcal{B})$ precisely enforces P over $\mathcal{A}^* \cup \mathcal{B}^*$ then it will revise any word $\alpha \in \mathcal{A}^*$ to its longest correct prefix wrt. P.

It might be the case, however, that a controller withholds or inserts some actions upon revising a word without changing its semantic meaning. For those types of controllers the notion of effective

enforcement has been introduced [21]. A controller that effectively enforces some property will rewrite any correct word to a similar word according to $\sim$.

Definition 5 (Effective Enforcement, [21]) *A controller* m *effectively enforces* P *wrt.* $\sim$ *iff for all* $\alpha \in \mathcal{A}^*$ *it holds that: (1)* $\mathsf{m}(\alpha) \in \mathsf{P}$ *and (2)* $\alpha \in \mathsf{P}$ *implies* $\alpha \sim \mathsf{m}(\alpha)$.

Example 5 (Ex. 2 cont., Effective Enforcement) *Let* m *be a controller over* $\mathcal{A}$ *such that for a given word any occurrence of* w *is revised to* ws *(i.e. a save is forced after each write action). We could argue that in our scenario saving the file multiple times between write operations equals saving the file once between write operations. This can be modelled by* $\sim$*. Then,* m *effectively enforces* P *wrt.* $\sim$*. Note that if we would take equality* $=$ *as similarity relation then* m *would neither precisely enforce* P *nor would it effectively enforce* P *wrt.* $=$.

Note that effective enforcement wrt. some property by a controller m does not require that all prefixes of correct words are rewritten by m to themselves as it was the case for perfect enforcement. Also note that if m precisely enforces P, then it also effectively enforces P up to any similarity relation. If m effectively enforces a safety property P wrt. $=$, then it also precisely enforces P. Properties for which violating words may not have a longest correct prefix cannot be effectively enforced by any controller [17].

3 Norms

Norms are a means to specify desirable behavior. Given a word representing a behavior, a norm might be violated multiple times. For instance norms that are represented as conditional obligations with deadlines (cf. [8, 4]) are violated each time that the deadline occurs and the obligation has not been satisfied since the last time the condition held. A controller that enforces a norm should either prevent violations, or sanction violations, which is known as regimentation or sanctioning, respectively.

For runtime control it is required that the violation of a norm should be detectable after a finite amount of actions. We therefore represent all violations of a norm as a set of words such that a violation occurs necessarily at the last action of those words, but possibly also earlier. A norm itself is represented as a tuple that contains the violations of the norm and the sanction that should be applied after a violation occurs, in case the norm's violations should be sanctioned.

We assume a global and fixed set of actions $\mathcal{S}$ that can be used as sanctions and use $\mathcal{A}_\mathcal{S}$ as shorthand for $\mathcal{A} \cup \mathcal{S}$. We also assume that the target system will not by itself try to execute sanctions. Hence if $\mathcal{A}$ are all actions that the target system may try to execute, then the set of possible sanctions $\mathcal{S}$ is disjoint from $\mathcal{A}$. Another consequence is that we define a norm's violations as a subset of $\mathcal{A}^*$. Finally we assume that for two different sanctions $s, s' \in \mathcal{S}$ there is no two words $\alpha, \alpha' \in \mathcal{A}_\mathcal{S}$ such that $\alpha s \alpha' \sim \alpha s' \alpha'$.

Definition 6 (Norm, η) *A norm is represented as a tuple* $\eta = (V, s)$ *where* $V \subseteq \mathcal{A}^*$ *is the set of violations and* $s \in \mathcal{S}$ *is a sanction. Furthermore if* $\alpha \in V$ *and* $\alpha \sim \alpha'$ *then also* $\alpha' \in V$ *for all* $\alpha, \alpha' \in \mathcal{A}^*$*. A word* $\alpha \in \mathcal{A}^*$ *is a* violation *of* η *iff* $\alpha \in V$*. Moreover,* α *violates* η *if there is a subword* $\alpha' \preceq \alpha$ *that is a violation of* η.

It is important to note that our representation of norms as a set of violating behaviors with a sanction is general and covers many other possible representations. For instance, norms for which violations are regular, in the sense that they can be specified by a formula (e.g. by using linear temporal logic), as well as norm for which violations are irregular are covered. For the irregular case, the violations of a norm might be specified by an enumeration of bad behaviors, e.g. a corpus of empirically collected bad behaviors or practices.

We stress that given our representation of norm violations, each norm violation is a word that violates the norm, but not every violating word is a norm violation. Next we need to define what it means for a word to be compliant with a norm. This is the case if the word does not violate the norm, or if each norm violation is immediately followed by the norm's sanction. Because sanctions can occur in compliant words we have that the set of compliant words for a norm is a subset of $\mathcal{A}^*_\mathcal{S}$. To define norm compliance we use $\alpha^{-\mathcal{S}}$ to refer to the word α' which equals α but in which all sanction actions from $\mathcal{S}$ are removed.

Definition 7 (Compliant words, P_η) *Let* $\eta = (V, s)$*. The set of* η*-compliant words* $\mathsf{P}_\eta \subseteq \mathcal{A}^*_\mathcal{S}$ *is the set of words* $\alpha \in \mathcal{A}^*_\mathcal{S}$ *such that* $\alpha^{-\mathcal{S}} \notin V$ *or* $\alpha[i{+}1] = s$ *for all* $i \in \{1, ..., |\alpha|{-}1\}$ *where* $\alpha^{-\mathcal{S}}_{..i} \in V$.

Note that aside from words with correctly sanctioned violations, P_η also includes all words $\alpha \in \mathcal{A}^*$ that do not violate a norm η. We observe a connection between norms and properties. For a norm η the set of compliant words P_η is a property, because: (1) ε contains no violation of a norm η and hence must be in P_η and (2) if a compliant word is similar to another compliant word, then both are compliant words, and hence both are in P_η.

Example 6 (Norm) *We assume that the set of sanctions is* $\mathcal{S} = \{u\}$ *where* u *stands for "undo the last write action that occurred". In other scenarios we may use more traditional sanctions such as fines or warnings. We will specify a norm that says that the file has to be saved between writes, and the sanction of which is the undo action. The set* $V \subseteq \mathcal{A}^*$ *contains all words* $\alpha \in \mathcal{A}^*$ *such that the final action is a write action and no save action has occurred since the last write action. Consider the norm* $\eta = (V, u)$*. The word* $\alpha = wswrws$ *violates* η*, because the prefix* $wswrw$ *of* α *is a violation of* η*. The* η*-compliant words* P_η *are those that do not violate* η*, such as* $wswrsws$*, or those where the sanction is applied after each violation, such as* $wswrwus$*. Note that the word* $wswrwus$ *still violates* η*, even though the sanction was applied.*

3.1 Regimenting Controller

A regimenting controller for a norm prevents norm violations. Such a controller halts the system execution if it is about to violate a norm.

Definition 8 (Regimenting Controller) *Let* η *be a norm and* m *be a controller over* $\mathcal{A}_\mathcal{S}$*.* m *is a regimenting controller for* η *iff for all* $\alpha \in \mathcal{A}^*$ *it holds that* $\mathsf{m}(\alpha) = \alpha'$ *where* α' *is the longest prefix of* α *that does not violate* η.

Example 7 (Ex. 6 Cont., Regimenting Controller) *Let* m_r *be a regimenting controller for the norm* η *from Example 6. We can have for example that* $\mathsf{m}_r(wswrws) = wswr$*. The norm's violation is prevented by blocking further execution when the norm is about to be violated by the third write action.*

We shall now establish a connection between regimenting controllers and precise enforcement. We first observe that for a norm η the set of η-compliant words P_η is not in general a safety property. Hence precisely enforcing P_η is not in general possible. Consider the norm $\eta = (V, u)$ from Example 6 and the word ww. As ww

contains an unsanctioned violation it is not in P_η. However, wwu is in P_η, as all violations are properly sanctioned. Hence, an incorrect word might be extended to a correct one showing that P_η is not a safety property. The subset of P_η that contains no norm violations is however always a safety property.

Proposition 1 *Let η be a norm and $\mathsf{P} \subseteq \mathsf{P}_\eta$ be the set of all words not violating η. Then, P is a safety property.*

Proof: *If a word $\alpha \in \mathcal{A}^*_\mathcal{S}$ violates η, then there is a prefix $\alpha' \preceq \alpha$ such that $\alpha' \in V$. It is impossible to add an extension to α such that α' is not a prefix anymore, hence any extension would be violating the norm as well and hence not be in P.*

A controller is a regimenting controller for a norm iff it precisely enforces the property that contains all words without norm violations. This means that we can apply security monitors [24]/truncation automata [21] for the regimentation of norms and make use of formal results of these techniques.

Proposition 2 *Let m be a controller over $\mathcal{A}_\mathcal{S}$, $\eta = (V, s)$ be a norm and $\mathsf{P} \subseteq \mathsf{P}_\eta$ be the set of all words not violating η. Then, m is a regimenting controller for η iff m precisely enforces P.*

Proof: *Let $\alpha \in \mathcal{A}^*_\mathcal{S}$ be an arbitrary word. By definition m can only revise a word α to its longest prefix $\alpha' \preceq \alpha$ that contains no violation of η. As α' has no violations of η we have that $\alpha' \in \mathsf{P}$, hence $\forall \alpha \in \mathcal{A}^*_\mathcal{S} : \mathsf{m}(\alpha) \in \mathsf{P}$, which is required for precise enforcement. If $\alpha \in \mathsf{P}$ then all prefixes of α are in P. If a word contains no norm violations then it is its own longest correct prefix given P, and hence is mapped to itself by m. This holds for all prefixes of α, therefore: for all $\alpha' \in \mathcal{A}^*_\mathcal{S}$ with $\alpha' \preceq \alpha$ it holds that $\mathsf{m}(\alpha') = \alpha'$.*

For the other direction, if m precisely enforces P then each word in P is mapped to itself, and hence is its own longest correct prefix. If a word is not in P then it is rewritten to its longest correct prefix, which in this case is the longest prefix such that η is not violated. This matches the definition of a regimenting controller.

3.2 Sanctioning Controller

A sanctioning controller for a norm revises a word by inserting a sanction after each violation of the norm. Hence it will make any word a norm compliant one, but does not prevent the norm violation like a regimenting controller. As before, we use $\alpha^{-\mathcal{S}}$ to refer to the result of removing all sanctions from α.

Definition 9 (Sanctioning Controller) *Let $\eta = (V, s)$ be a norm and m be a controller over $\mathcal{A}_\mathcal{S}$. m is a sanctioning controller for η iff for all $\alpha \in \mathcal{A}^*_\mathcal{S}$ it holds that $\mathsf{m}(\alpha) \in \mathsf{P}_\eta$ and $\mathsf{m}(\alpha)^{-\mathcal{S}} = \alpha^{-\mathcal{S}}$.*

Example 8 (Ex. 6 Cont., Sanctioning Controller) *Let η be the norm from Example 6 and m_s be a sanctioning controller for η. A revision of m_s is $\mathsf{m}_s(wswrws) = wswrwus$. The norm's violation is sanctioned by undoing the last write action when the violation occurred.*

We shall now establish the connection between sanctioning controllers and effective enforcement. Note first that a controller that effectively enforces P_η for some norm η is not necessarily a sanctioning controller. If m effectively enforces P_η then it is allowed that for a word $\alpha \notin \mathsf{P}_\eta$ it holds that $\mathsf{m}(\alpha) = \alpha'$ where α' is the longest correct prefix of α in P_η. However, the definition of sanctioning controllers requires that the controller injects sanctions which m does not do. A sanctioning controller can possibly duplicate sanctions if they already occur in an input word. If a norm η has a sanction s which may be duplicated in any word in which s occurs without changing the word in a meaningful way wrt. $\sim$, then a sanctioning controller for a norm η effectively enforces P_η. This means that edit automata [21] can be used to implement sanctioning controllers and we can make use of results in that area to analyze sanctioning controllers.

Proposition 3 *Let m be a controller over $\mathcal{A}_\mathcal{S}$, η be a norm, and $\sim$ be the identity relation with the constraint that for all $\alpha, \alpha' \in \mathcal{A}^*_\mathcal{S}$ and all $s \in \mathcal{S}$ it holds that $\alpha ss \alpha' \sim \alpha s \alpha'$. If m is a sanctioning controller for η then m effectively enforces P_η.*

Proof: *Let $\alpha \in \mathcal{A}^*_\mathcal{S}$ be an arbitrary word and $\mathsf{m}(\alpha) = \alpha'$. The definition of a sanctioning controller ensures that each violation in α is followed by s. Hence, for all $\alpha \in \mathcal{A}^*_\mathcal{S}$ it holds that $\mathsf{m}(\alpha) \in \mathsf{P}_\eta$; the first constraint of effective enforcement. Second, if $\alpha \in \mathsf{P}_\eta$ then any occurring violation of η in α is sanctioned. m inserts sanctions after each violation, even if a sanction already follows, hence duplications may occur in $\mathsf{m}(\alpha)$ if sanctions already occur in α. But the duplication of a sanction was assumed to not change the word in a meaningful way wrt. $\sim$. Hence for all $\alpha \in \mathsf{P}_\eta$ we have that $\mathsf{m}(\alpha) \sim \alpha$; the second constraint of effective enforcement.*

4 Controller Automata

We introduce controller automata which are a formal tool to model runtime controllers in more detail. A controller automaton is essentially the same as an edit automaton introduced in [21]. The difference is of a syntactic rather than conceptual nature: they are equally expressive. Controller automata are labeled transition systems over some input alphabet $\mathcal{A}$ and an output alphabet $\mathcal{B}$. Such an automaton makes a transition from one state to another state by reviewing an action and performing a revision. A *revision* is given by a/X where $a \in \mathcal{A} \cup \mathcal{B} \cup \{\varepsilon\}$ is an output action (or the empty word) to which an input action is revised and $X \in \{A, I, S, L\}$ is the name of the revision operation, where A stands for allow, I for insert, S for suppress and L for loop. Suppose some controller automaton is given the input word $\alpha = a_1 a_2 \dots a_k$. It reviews the input from left to right starting at a_1. Then upon reviewing a_i, $i \in \{1, ..., k\}$: a_i/A is read as "a_i is allowed, continue with $\alpha_{i+1..}$", ε/S is read as "a_i is suppressed, continue with $\alpha_{i+1..}$", a'/I is read as "a' is inserted in the output word, keep reviewing $\alpha_{i..}$", and ε/L is read as "do nothing, keep reviewing $\alpha_{i..}$". The minor difference to edit automata is the loop revision operator (as first class citizen in the revision set). This is required for the distributed setting in the next section. We also omit the "halt" revision present in edit automata. A halt operation can be modeled by a special sink state that suppresses any action with a reflexive transition. We recall the definition of edit automata [21] with minor adjustments required for our setting. We require that for each possible input word the controller automaton halts at some point, which guarantees that the output word is always finite. This requirement is met if there is no transition in the controller automaton such that it can infinitely apply the insert revision. In practice this requirement is rarely limiting as control mechanisms are meant to modify executions, and not take over the flow of execution.

Definition 10 (Controller Automaton, c) *A* controller automaton *is a tuple $\mathsf{c} = (\mathcal{A}, \mathcal{B}, Q, q_0, \delta)$ consisting of an input alphabet $\mathcal{A}$ and an output alphabet $\mathcal{B}$, a countable set of (control) states Q, an initial state $q_0 \in Q$ and a transition function $\delta : Q \times \mathcal{A} \to \mathsf{Rev} \times Q$ where $\mathsf{Rev} = \{a/X \mid a \in \mathcal{A} \cup \mathcal{B},\ X \in \{A, I\}\} \cup \{\varepsilon/S, \varepsilon/L\}$*

is the set of revisions. Moreover, we require that there is no infinite sequence of control states $q_1 q_2, \ldots$ such that $\delta(q_1, a) = (\alpha/I, q_2)$, $\delta(q_2, a) = (\alpha'/I, q_3)$, etc.

In the following we assume, if not said otherwise, that a controller $\mathsf{c} = (\mathcal{A}, \mathcal{B}, Q, q_0, \delta)$ is given. The operational semantics describes how the automaton behaves on an input word.

Definition 11 (Operational semantics) *A* configuration *of* c *is a tuple $(\alpha, q) \in \mathcal{A}^* \times Q$ where α represents the input word that remains to be reviewed and q is the current control state. The operational semantics is defined by the following transition rule:*

$$\frac{\delta(q, a) = (r, q')}{(a\alpha, q) \xrightarrow{r}_{\mathsf{c}} (\alpha', q')} \qquad \textit{(Controller Transition)}$$

where $\alpha' = \alpha$ if $r = a/A$ or $r = \varepsilon/S$, otherwise $\alpha' = a\alpha$. We write $\mathsf{ctrl}_{\mathsf{c}}(\alpha) = \alpha_1 \alpha_2 \ldots \alpha_n$ iff controller c can make the transitions $(\alpha, q_0) \xrightarrow{\alpha_1/X_1}_{\mathsf{c}} (\alpha', q_1) \xrightarrow{\alpha_2/X_2}_{\mathsf{c}} \ldots \xrightarrow{\alpha_n/X_n}_{\mathsf{c}} (\varepsilon, q_n)$. Often, we also identify $\mathsf{ctrl}_{\mathsf{c}}$ with c.

A transition $(a\alpha, q) \xrightarrow{r}_{\mathsf{c}} (\alpha', q')$ represents that in state q, when action a is being reviewed, a transition to state q' takes place whilst revision r is executed, and the automaton continues reviewing the input α'. Note that for each $\alpha \in \mathcal{A}^*$ there is exactly one possible output word provided by $\mathsf{ctrl}_{\mathsf{c}}(\alpha)$.

Proposition 4 *For every controller automaton c, we have that $\mathsf{ctrl}_{\mathsf{c}} : \mathcal{A}^* \to \mathcal{B}^*$ is a controller.*

Given this result we will also say that "c enforces…" when meaning that "$\mathsf{ctrl}_{\mathsf{c}}$ enforces…", etc.

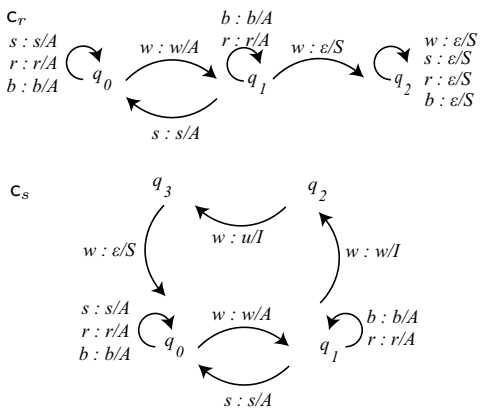

Figure 1. Top: controller automaton implementing m_r from Example 7, bottom: controller automaton implementing m_s from Example 8. A transition from $q \xrightarrow{a:\alpha/X} q'$ indicates that upon reviewing action a in state q the controller transitions to q' whilst executing revision α/X.

Example 9 (Ex. 8 Cont., Controller automata) *In Figure 1 two example controller automata c_r (top) c_s (bottom) are shown. We have that $\mathsf{ctrl}_{\mathsf{c}_r}$ and $\mathsf{ctrl}_{\mathsf{c}_s}$ are the controllers m_r and m_s from Examples 7 and 8, respectively. According to definition 10 states q_2 and q_3 in the automaton c_s should have transitions for each action. We omitted some because a transition that inserts an action upon reviewing w will not consume w, hence the next transition must be triggered by w again. We also omitted transitions for u as we assume that the target system will not produce this action.*

The connection between regimenting and sanctioning controllers for norms and controller automata is that any regimenting or sanctioning controller can be implemented by a controller automaton.

Proposition 5 *Let η be a norm and controller m be a regimenting or sanctioning controller for η. Then, there is a controller automaton c such that $\mathsf{ctrl}_{\mathsf{c}} = \mathsf{m}$.*

Proof (sketch): *Recall from Proposition 2 that a regimenting controller for a norm η precisely enforces the set of words $\mathsf{P} \subseteq \mathsf{P}_\eta$ that do not violate η. A controller automaton is as expressive as an edit automaton. For edit automata it is shown that they can specify precisely enforcing controllers [21]. Therefore a regimenting controller for P can be implemented by an edit automaton and hence by a controller automaton. A controller automaton $\mathsf{c} = (\mathcal{A}_\mathcal{S}, \mathcal{A}_\mathcal{S}, Q, q_0, \delta)$ such that $\mathsf{ctrl}_{\mathsf{c}} = \mathsf{m}$ for a sanctioning controller m for $\eta = (V, s)$ can be constructed as follows: (1) for each word $\alpha \in \mathcal{A}^*_\mathcal{S}$ assign a new state q in Q and for ε that state is q_0, (2) for each action $a \in \mathcal{A}_\mathcal{S}$ and word $\alpha \in \mathcal{A}^*_\mathcal{S}$ let q and q' be the states belonging to α and αa respectively, define $\delta(q, a) = (a/A, q')$ if $\alpha a \notin V$, otherwise make two new states q_1, q_2, add them to Q, and define: $\delta(q, a) = (a/I, q_1)$, $\delta(q_1, a) = (s/I, q_2)$, $\delta(q_2, a) = (\varepsilon/S, q')$. Note that by this construction Q is countably infinite.*

5 Distributed Controllers

For many applications it is required that multiple norms are enforced. This can be achieved by deploying a distributed controller that consists of a set of concurrently applied individual controllers that collectively enforce norms. In this section we discuss collaborative automata that represent distributed controllers. It is important to note that a collaborative automaton is not an entity that has to be implemented alongside the individual controllers, but it is merely a representation of how the individual controllers are synchronized. Also, we assume here the extreme case where there is full synchronization among controllers and the target system. This is not always a requirement however when one implements the controllers that make up a collaborative automaton. It must be ensured, however, that the output of the collaborative automaton contains only norm compliant words, or restricts input words to words without any norm violations, for any norm. If multiple controllers, each of which is a sanctioning or regimenting controller for one of the norms, are applied concurrently on the same target system, then the compatibility of controllers must be ensured. For instance, when reviewing some action, some controllers may propose to suppress that action whilst other controllers propose to allow that action. A conflict resolution mechanism should in those cases decide which of the proposed revisions is applied. Note that this conflict resolution is on the level of action revisions, and not on a normative level. It is possible to have two controllers as part of a collaborative controller which enforce norms that cannot both be complied with. In case of regimentation it would then be ensured that at some point the controller halts the target system. In case of sanctioning it would mean that there will be a point where the sanction is applied. For a discussion on normative conflict, see for instance the work by Vasconcelos et al. [29].

Consider again the controller automata from Example 9 and Figure 1. If they simultaneously review ww then the revisions that the top controller automaton c_r executes are first w/A and then ε/S, whereas the bottom controller automaton c_s executes w/A, w/I, u/I and then ε/S. Both controller automata agree on the initial allow revision (w/A) but then execute different revisions. For this purpose, we introduce a *selection function* that decides which controller

automaton's revisions are performed. The main challenge when constructing a collaborative automaton is to define an appropriate selection function. Also note that the number of revisions for controller automata may not be equal as in the example above. This happens when some of the controllers perform insert revisions and thus do not move to the next input action, whilst other automata may perform allow or suppress revisions so that they do move on to next action. We make use of loop revisions to maintain synchronization. If some automata can perform an insert revision whilst others do not, then those others have to loop until all automata are ready with insertions and can allow or suppress the action that is being reviewed. In our example, the top controller automaton c_r should be forced to loop when the second action w is under review until c_s is at state q_3 and, just like c_r, is ready to process w by an allow or suppress revision.

5.1 Collaborative Automaton

A collaborative automaton models the collaboration between a set of concurrent controller automata. A collaborative automaton has a state space and transition function that given a state and action returns a label and next state. A transition is labeled by a vector of revisions, one for each controller automaton. A selection function can pick a revision from this vector for each transition. This selection can be interpreted as the result of coordination between controllers to decide upon a revision. The state space and transition function of a collaborative automaton are constructed from the individual controller automata. The state space is essentially the Cartesian product of the state spaces of the controller automata where we need to duplicate the local states of each controller automaton to allow them to loop if necessary. The duplicate of a state q is denoted by $\widehat{q}$. In the following, for a state $x \in \{q, \widehat{q}\}$ we write $\overline{x}$ to refer to q, i.e., the overline removes the hat annotation. For a set of controller automata, the collaborative state is hence a snapshot of the states in which each controller automaton is at a certain moment in time. If a controller automaton's state in a collaborative state is $\widehat{q}$ then this can be interpreted as that the controller automaton is 'on hold'. The combination of all initial states of the controller automata is the initial state of the collaborative automaton.

We say a controller automaton proposes a certain revision given a collaborative state and action when, given the automaton's local state in the collaborative state and the action, the controller automaton can make a transition with that revision as a label. We say that the revisions in a label from a collaborative transition are assigned to the controller automata. However, in an application these revisions are the result of synchronization between controller applications. A transition label from one state of the collaborative automaton to the next for a given action is constructed as follows:

1. If there is a controller automaton that proposes an insert, then the collaborative automaton assigns to each controller automaton that proposes an insert their proposal. The other controller automata, which either propose an allow or suppress revision, are assigned loop revisions by the collaborative automaton.
2. If no controller automaton proposes an insert, then all controller automata propose either an allow or suppress revision. The collaborative automaton assigns to each controller automaton the revisions that they themselves propose.

The next state after a transition is determined by the label. Assume some given action. If a controller automaton is in state q in some collaborative state, and its assigned revisions given the collaborative state and action is the loop revision, then its next state becomes $\widehat{q}$ in the next collaborative state. If, however, another revision was assigned to the controller automaton, then it must be a label of a transition that the controller automaton can make upon reading the action. The resulting state of that transition becomes the state of the controller automaton in the next collaborative state. We give the formal definition of a collaborative automaton. Recall that Rev is the set of possible revisions.

Definition 12 (Collaborative Automaton, C) *Let* $\mathsf{M} = \{\mathsf{c}_1, ..., \mathsf{c}_k\}$ *be a set of controller automata such that* $\mathsf{c}_i = (\mathcal{A}, \mathcal{B}, Q^i, q_0^i, \delta^i)$ *and* $\widehat{Q}^i = Q^i \cup \{\widehat{q} \mid q \in Q^i\}$. *A collaborative automaton* C *over* M *is a labeled transition system* $\mathsf{C} = (\mathcal{A}, \mathcal{B}, \mathsf{Q}, \mathsf{q}_0, \Delta, \sigma)$, *where* $\mathsf{Q} = \widehat{Q}^1 \times ... \times \widehat{Q}^k$ *is the set of collaborative states,* $\mathsf{q}_0 = (q_0^1, ..., q_0^k)$ *is the initial collaborative state, and* $\Delta : \mathsf{Q} \times \mathcal{A} \to \mathsf{Rev}^k \times \mathsf{Q}$ *is the transition function defined as follows.* $\Delta((x_1, ..., x_k), a) = ((r_1, ..., r_k), (y_1, ..., y_k))$ *if, and only if, it holds that:*

1. *If there is an* $i \in \{1, ..., k\}$ *such that* $\delta^i(\overline{x_i}, a) = (a'/I, q')$ *then for all* $j \in \{1, ..., k\}$ *it holds that:*

$$\begin{cases} r_j = a'/I, y_j = q' & \text{if } \delta^j(\overline{x_j}, a) = (a'/I, q') \\ r_j = \varepsilon/L, y_j = \widehat{\overline{x_j}} & \text{otherwise;} \end{cases}$$

2. *otherwise,* $\delta^j(\overline{x_j}, a) = (r_j, y_j)$, *for all* $j \in \{1, ..., k\}$.

Finally, σ *is a function* $\mathsf{Q} \times \mathcal{A} \to \mathsf{Rev}$ *such that* $\sigma(\mathsf{q}, a) \in \{r_1, ..., r_k\}$ *where* $\Delta(\mathsf{q}, a) = ((r_1, ..., r_k), \mathsf{q}')$. *The function is called the* selection function *of* C.

Each transition step of the collaborative automaton is labeled with a vector of revisions. We note that a collaborative automaton is completely specified by M apart from the selection function. In the following we assume that $\mathsf{C} = (\mathcal{A}, \mathcal{B}, \mathsf{Q}, \mathsf{q}_0, \Delta, \sigma)$ is given as in the definition above.

Definition 13 (Operational Semantics) *A* configuration *of* C *is a tuple* $(\alpha, \mathsf{q}) \in \mathcal{A}^* \times \mathsf{Q}$ *where* $\alpha \in \mathcal{A}^*$ *is the input word that remains to be reviewed and* q *is the current state of* C. *The operational semantics of a collaborative automaton is defined by the following transition rule:*

$$\frac{\Delta(\mathsf{q}, a) = ((r_1, \ldots, r_k), \mathsf{q}')}{(a\alpha, \mathsf{q}) \xrightarrow{\sigma(\mathsf{q}, a)}_{\mathsf{C}} (\alpha', \mathsf{q}')} \qquad \textit{(Col. Transition)}$$

where $\alpha' = a\alpha$ *if* $\sigma(\mathsf{q}, a) \in \{a'/I \mid a' \in \mathcal{A} \cup \mathcal{B}\} \cup \{\varepsilon/L\}$, *and* $\alpha' = \alpha$ *otherwise. As before, we write* $\mathsf{ctrl}_\mathsf{C}(\alpha) = \alpha_1\alpha_2 \ldots \alpha_n$ *iff* C *can make the transitions* $(\alpha, q_0) \xrightarrow{\alpha_1/X_1}_{\mathsf{C}} (\alpha', q_1) \xrightarrow{\alpha_2/X_2}_{\mathsf{C}} \ldots \xrightarrow{\alpha_n/X_n}_{\mathsf{C}} (\varepsilon, q_n)$. *Again, we often identify* ctrl_C *with* C.

Proposition 6 *For every collaborative automaton* C, *we have that* $\mathsf{ctrl}_\mathsf{C} : \mathcal{A}^* \to \mathcal{B}^*$ *is a controller.*

Example 10 (Ex. 9 Cont., Collaborative Automaton) *In Figure 2 two controller automata* c_1 *and* c_2 *with* $\mathsf{c}_i = (\mathcal{A}_\mathcal{S}, \mathcal{A}_\mathcal{S}, Q^i, q_0^i, \delta^i)$, *are shown. Controller* c_1 *is controller* c_s *from Example 9 which is a sanctioning controller for* $\eta_1 = (V_1, u)$. *Controller automaton* c_2 *is an implementation of a sanctioning controller for the norm* $\eta_2 = (V_2, u)$ *where* V_2 *contains all words where a write action is not immediately preceded by a backup action. In Figure 2 there is also an example collaborative automaton* $\mathsf{C} = (\mathcal{A}_\mathcal{S}, \mathcal{A}_\mathcal{S}, \mathsf{Q}, \mathsf{q}_0, \Delta, \sigma)$ *over*

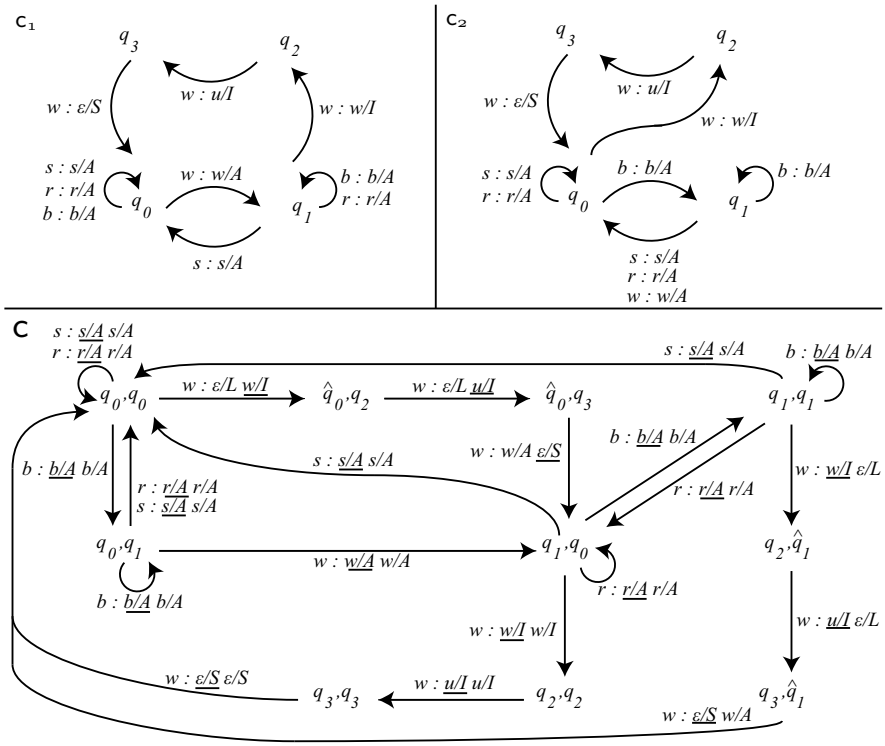

Figure 2. Example controller automata and collaborative automaton. Description in Example 10.

$\{\mathsf{c}_1, \mathsf{c}_2\}$. *For a collaborative transition we use* $(x_1, x_2) \xrightarrow{a:r_1 r_2} (y_1, y_2)$ *to indicate that a controller* c_i *can make a transition from* x_i *to* y_i *when reviewing an action* a *whilst executing* r_i. *It also indicates that* $\Delta((x_1, x_2), a) = ((r_1, r_2), (y_1, y_2))$. *The underlined revision indicates the selection made by the selection function* σ. *E.g. from the figure we can see that* $\sigma((q_0, q_0), w) = w/I$. *We again omit some transitions like in Example 9.*

Observe that for any input word the output is both η_1 *compliant and* η_2 *compliant.* η_1 *requires that a write action is followed by the undo action if no save occurred since the last write action.* η_2 *requires that each write action which was not immediately preceded by a backup is followed by the undo action. So for instance if* $\alpha = w$ *then immediately* η_2 *is violated. In* C *we can see that reviewing* α *will immediately go from the initial state through three other states such that during those transitions* c_2 *sanctions the violation and* c_1 *does nothing, but ends in a state as if it has allowed* w. *It can also be seen that loops and duplicate states are required for synchronization. We can see that* $\mathsf{ctrl}_{\mathsf{C}}(w) = wu$. *If* $\alpha = ww$, *then after the second* w *both controllers detect a norm violation and want to insert the sanction. But it suffices to only once insert the sanction, which is what happens in the collaborative automaton, i.e.* $\mathsf{ctrl}_{\mathsf{C}}(ww) = wuwu$. *Note that if the sanctions of the controllers would differ, then in this particular case we had to choose between the sanctions in the transition between state* (q_1, q_0) *and* (q_2, q_2), *which may have resulted in non-compliance of the output given the input* ww.

5.2 Collaborative Automata for Norms

We want to investigate whether a given set of norm enforcing controllers can be concurrently applied. We do this by showing the existence of a collaborative automaton, since these capture the possible synchronizations among individual controllers. The existence of a collaborative automaton does not specify a communication protocol to ensure norm compliant behavior, but does show that such a protocol exists and hints on the construction of the protocol. We can combine controller automata of regimenting controllers such that the collaborative automaton will prevent any norm violation. For this purpose we first provide a proposition that explains the connection between collaborative automata and precise enforcement.

We follow [17] and consider only controllers that rewrite an incorrect word to its longest correct prefix wrt. the property it enforces. In [21] it is shown that precisely enforcing controllers can be modeled with automata that do not use insert revisions. Thus, we can assume that controller automata that specify precisely enforcing controllers do not use insert revisions. Under these constraints, we can determine a collaborative automaton, by finding an appropriate selection function, that precisely enforces the intersection of a set of precisely enforced properties.

Proposition 7 *Let* $\mathsf{M} = \{\mathsf{c}_1, ..., \mathsf{c}_k\}$ *be a set of controller automata that do not perform insert revisions where* c_i *precisely enforces property* P_i *such that if* $\alpha \notin \mathsf{P}_i$ *then* c_i *rewrites* α *to its longest correct prefix wrt.* P_i. *Then, there exists a collaborative automaton* C *over* M *such that* $\mathsf{ctrl}_{\mathsf{C}}$ *precisely enforces* $\mathsf{P}_\cap = \bigcap_{i \in \{1,...,k\}} \mathsf{P}_i$.

Proof (sketch): *Let* $\mathsf{C} = (\mathcal{A}, \mathcal{B}, \mathsf{Q}, \mathsf{q}_0, \Delta, \sigma)$ *be the collaborative automaton. We define the following selection function* σ: $\sigma(\mathsf{q}, a) = \begin{cases} \varepsilon/S & \text{if } \exists i \in \{1, \ldots, k\} : r_i = \varepsilon/S \\ a/A & \text{otherwise} \end{cases}$
where $\Delta(\mathsf{q}, a) = ((r_1, \ldots, r_k), \mathsf{q}')$.

For precise enforcement we have to show that for each $\alpha \in \mathcal{A}$ *we have that* $\mathsf{ctrl}_{\mathsf{C}}(\alpha) \in \mathsf{P}_\cap$ *and if* $\alpha \in \mathsf{P}_\cap$ *then we have for each* $\alpha' \preceq \alpha$ *that it holds that* $\mathsf{ctrl}_{\mathsf{C}}(\alpha') = \alpha'$.

Now, suppose $\alpha \in \mathcal{A}^*$. We have that $\mathsf{c}_i(\alpha) = \alpha^i \in \mathsf{P}_i$ for all i. Let $\alpha' = a'_1 \dots a'_j$ be the longest common prefix of all the $\mathsf{c}_i(\alpha)$. By precise enforcement, $\alpha' \in \mathsf{P}_\cap$. On input α no controller automaton can suppress for the first j transitions. If $\alpha = \alpha'$ then also $\mathsf{ctrl}_\mathsf{C}(\alpha) = \alpha' \in \mathsf{P}_\cap$, because no suppression occurs. If $\alpha' \prec \alpha$ then some controller automaton must suppress the next input action a_{j+1}. Moreover, this automaton will keep on suppressing actions from that moment on, and therefore so will C, showing that $\mathsf{ctrl}_\mathsf{C}(\alpha) = \alpha' \in \mathsf{P}_\cap$. Secondly, suppose that $\alpha \in \mathsf{P}_\cap$. Then for each i, also $\alpha \in \mathsf{P}_i$ and thus for all $\alpha' \preceq \alpha$ it holds that $\mathsf{c}_i(\alpha') = \alpha'$; in particular, c_i allows all actions of α'. Thus, by definition of the selection function, $\mathsf{ctrl}_\mathsf{C}(\alpha') = \alpha'$.

From Proposition 7 in combination with Proposition 2 it follows that a set of regimenting controllers $\{\mathsf{c}_1, ..., \mathsf{c}_k\}$ for the norms $\eta_1 = (V_1, s_1), \dots, \eta_k = (V_k, s_k)$ can be combined to a collaborative automaton that prevents any violation of a norm η_i, $i \in \{1, ..., k\}$. Note that this is also the same as stating that the collaborative automaton specifies a regimenting controller for a norm $(V_\cup, s)$, where $V_\cup = \bigcup_{i\in\{1,\dots,k\}} V_i$ and $s \in \mathcal{S}$ is an arbitrary sanction (because sanctions play no role in regimentation). For simplicity we again assume that the controllers do not use insert revisions.

Theorem 1 *Let $\mathsf{M} = \{\mathsf{c}_1, ..., \mathsf{c}_k\}$ be a set of controller automata over $\mathcal{A}_\mathcal{S}$ which implement regimenting controllers for norms $\eta_1 = (V_1, s_1), ..., \eta_k = (V_k, s_k)$ and do not use insert revisions, and $\mathsf{P}_\cap = \bigcap_{i\in\{1,\dots,k\}} \mathsf{P}_i$, where $\mathsf{P}_i \subseteq \mathsf{P}_{\eta_i}$ are all words that do not violate η_i. Then, there exists a collaborative automaton C over M such that $\mathsf{ctrl}_\mathsf{C}(\alpha) \in \mathsf{P}_\cap$ for each $\alpha \in \mathcal{A}^*_\mathcal{S}$.*

Proof (sketch): *Recall from Proposition 7 that each controller c_i is precisely enforcing the property P_i. A regimenting controller c_i also rewrites a word $\alpha \in \mathcal{A}^*_\mathcal{S}$ to its longest correct prefix given P_i.*

Therefore $\mathsf{M} = \{\mathsf{c}_1, ..., \mathsf{c}_k\}$ is a set of controller automata that do not perform insert revisions where c_i precisely enforces property P_i such that if $\alpha \notin \mathsf{P}_i$ then c_i rewrites α to its longest correct prefix wrt. P_i. Following Proposition 7 there exists a collaborative automaton C over M such that ctrl_C precisely enforces $\mathsf{P}_\cap$. Hence for that controller C we have $\mathsf{ctrl}_\mathsf{C}(\alpha) \in \mathsf{P}_\cap$ for each $\alpha \in \mathcal{A}^*_\mathcal{S}$.

For a set of sanctioning controllers we may run into a conflict if two controllers propose to insert a different sanction at the same time. The selection function can only select one of those sanctions. This issue will not occur if for any two sanctioning controllers for norms $\eta_1 = (V_1, s_1)$ and $\eta_2 = (V_2, s_2)$ there is no situation where they both propose to insert a sanction (i.e. $V_1 \cap V_2 = \emptyset$) or if the sanctions are the same. The latter case may occur if for instance the same sanction is used for all norms, such as a warning. Also, one can model different sanction procedures with the same symbol, and only use different symbols if the procedures are incompatible. If no two different sanction actions can be inserted concurrently, then a collaborative automaton can be constructed such that each input word is revised to a η_i compliant word for each $i \in \{1, ..., k\}$.

Theorem 2 *Assume that $\sim$ is the identity relation with the constraint that $\forall s \in \mathcal{S}, \alpha \in \mathcal{A}^*_\mathcal{S}, \alpha' \in \mathcal{A}^*_\mathcal{S}$: $\alpha s \alpha' \sim \alpha s s \alpha'$. Let $\mathsf{M} = \{\mathsf{c}_1, ..., \mathsf{c}_k\}$ be a set of controller automata over $\mathcal{A}_\mathcal{S}$ which implement sanctioning controllers for norms $\eta_1 = (V_1, s_1), ..., \eta_k = (V_k, s_k)$ and $\mathsf{P}_\cap = \bigcap_{i\in\{1,\dots,k\}} \mathsf{P}_{\eta_i}$. Then, there exists a collaborative automaton C over M such that $\mathsf{ctr}_\mathsf{C}(\alpha) \in \mathsf{P}_\cap$ for each $\alpha \in \mathcal{A}^*_\mathcal{S}$ iff for each $i, j \in \{1, ..., k\}$ if $V_i \cap V_j \neq \emptyset$ then $s_i = s_j$.*

Proof (sketch): *For simplicity we assume that a controller automaton uses a/A if possible and not the equivalent revisions a/I followed by ε/S. In that case, note that the controller automata will only propose allow revisions, unless a violation is detected upon reviewing an action a, in which case a word of revisions equivalent to a/I, s/I, ε/S is executed, where $s \in \mathcal{S}$ is a sanction. Under this assumption let $\mathsf{C} = (\mathcal{A}_\mathcal{S}, \mathcal{A}_\mathcal{S}, \mathsf{Q}, \mathsf{q}_0, \Delta, \sigma)$ be the collaborative automaton. We define the following selection function σ:*

$$\sigma(\mathsf{q}, a) = \begin{cases} \varepsilon/S & \text{if } \exists i \in \{1, ..., k\} : r_i = \varepsilon/S \\ a'/I & \text{if } \exists i \in \{1, ..., k\} : r_i = a'/I \\ a/A & \text{otherwise} \end{cases}$$

where $\Delta(a, \mathsf{q}) = ((r_1, ..., r_k), \mathsf{q}')$.

Consider an arbitrary word $\alpha \in \mathsf{P}_\cap$ (such as ε), $\mathsf{q} \in \mathsf{Q}$ such that q is reached after reviewing α, and action $a \in \mathcal{A}_\mathcal{S}$. If $\alpha a \in \mathsf{P}_\cap$ then $\alpha a \in \mathsf{P}_{\eta_i}$ for each $i \in \{1, ..., k\}$, and hence each controller will allow the action. Therefore $\sigma(\alpha, a) = a/A$ and $\mathsf{ctrl}_\mathsf{C}(\alpha a) = \alpha a \in \mathsf{P}_\cap$. If $\alpha a \notin \mathsf{P}_\cap$ then, given our assumptions, for each $i \in \{1, ..., k\}$ such that $\alpha a \notin \mathsf{P}_{\eta_i}$ we know that c_i will sanction the violation with the same sanction $s \in \mathcal{A}_\mathcal{S}$. Therefore $\alpha a s \in \mathsf{P}_\cap$. These controllers will all first insert a, the others will have to loop. Hence $\sigma(\mathsf{q}, a) = a/I$. Then, all these controllers will insert s and the others will loop, so $\sigma(\mathsf{q}', a) = s/I$, finally, all the controllers that proposed inserting the action will now suppress a and the others will propose to allow a. Therefore $\sigma(\mathsf{q}'', a) = \varepsilon/S$ and $\mathsf{ctrl}_\mathsf{C}(\alpha a) = \alpha a s \in \mathsf{P}_\cap$. Hence for any word $\alpha \in \mathcal{A}^*_\mathcal{S}$ if α violates some η_i then in $\mathsf{ctrl}_\mathsf{C}(\alpha)$ the violation will be followed by s_i. Therefore, $\mathsf{ctrl}_\mathsf{C}(\alpha) \in \mathsf{P}_\cap$ for any word $\alpha \in \mathcal{A}^*_\mathcal{S}$.

6 Concluding Remarks

Norm enforcement requires an enforcing controller to influence a target system's behavior. Enforcement through regimenation can halt target systems, and sanctioning can inject sanction actions. Our first contribution in this paper is an analysis of how regimentation and sanctioning relate to work on runtime control and verification. This allows us to analyze the runtime enforcement of norms. Our second contribution in this paper is a framework to specify the concurrent application of a set of controllers. We investigated the conditions under which the norm enforcement of a set of norms is possible.

Work on team automata [15, 26] to enforce properties distributively [27] relates to our collaborative automata. In particular in [30] edit automata are combined into a team edit automata. In this framework it is not possible for individual controllers to loop, as they do in our framework. How the team edit automaton's transition function can be constructed is not specified in [30], neither is a formal analysis given. In [19] a collaborative monitoring system, called a service automata framework, is described. The differences with our framework are that their controllers have an explicit local view of the target system, controllers are required to unanimously react to events through allow/suppress/insert, and interaction consists of sharing observations and delegating revision decisions. It is not analyzed how separately developed service automata can be combined into a service automata framework.

We are currently working on the translation of theoretical work on runtime norm enforcement to practical programming frameworks for implementations (cf. [12]). One of the challenges that remains is to develop a methodology to exploit the structure of a specific collaborative automaton in order to create an efficient communication protocol. For instance in the collaborative automaton from Example 9 c_2 only needs to execute its proposed revisions if both controllers are in state q_0 and the next action to review is w. Hence no full synchronization of the controllers is necessary, which is desirable.

REFERENCES

[1] Thomas Ågotnes, Wiebe van der Hoek, and Michael Wooldridge, 'Normative system games', in *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '07)*, pp. 1–8, New York, NY, USA, (2007). ACM.

[2] Natasha Alechina, Nils Bulling, Mehdi Dastani, and Brian Logan, 'Practical run-time norm enforcement with bounded lookahead', in *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015*, pp. 443–451, (2015).

[3] Natasha Alechina, Mehdi Dastani, and Brian Logan, 'Reasoning about normative update', in *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, (2013).

[4] Natasha Alechina, Mehdi Dastani, and Brian Logan, 'Reasoning about normative update', in *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pp. 20–26. AAAI Press, (2013).

[5] Natasha Alechina, Mehdi Dastani, and Brian Logan, 'Norm approximation for imperfect monitors', in *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '14, Paris, France, May 5-9, 2014*, pp. 117–124, (2014).

[6] Guido Boella and Leendert van der Torre, 'Regulative and constitutive norms in normative multiagent systems', in *Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR'04)*, pp. 255–266, (2004).

[7] Olivier Boissier, Rafael H. Bordini, Jomi Fred Hübner, Alessandro Ricci, and Andrea Santi, 'Multi-agent oriented programming with jacamo', *Science of Computer Programming*, **78**(6), 747–761, (2013).

[8] Jan Broersen, Frank Dignum, Virginia Dignum, and John-Jules Ch Meyer, 'Designing a deontic logic of deadlines', in *Deontic Logic in Computer Science*, 43–56, Springer, (2004).

[9] Christos G. Cassandras and Stephane Lafortune, *Introduction to Discrete Event Systems*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[10] Edward Chang, Zohar Manna, and Amir Pnueli, *The safety-progress classification*, Springer, 1993.

[11] Mehdi Dastani, John-Jules Ch. Meyer, and Davide Grossi, 'A logic for normative multi-agent programs', *Journal of Logic and Computation*, **23**(2), 335–354, (2013).

[12] Mehdi Dastani and Bas Testerink, 'From multi-agent programming to object oriented design patterns', in *International Workshop on Engineering Multi-Agent Systems*, pp. 204–226. Springer, (2014).

[13] Mehdi Dastani, Nick AM Tinnemeier, and John-Jules Ch Meyer, 'A programming language for normative multi-agent systems', *Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, 397–417, (2009).

[14] Frank Dignum, 'Autonomous agents with norms', *Artificial Intelligence and Law*, **7**(1), 69–79, (1999).

[15] Clarence Ellis, 'Team automata for groupware systems', in *Proceedings of the international ACM SIGGROUP conference on Supporting group work: the integration challenge*, pp. 415–424. ACM, (1997).

[16] Marc Esteva, Julian Padget, and Carles Sierra, 'Formalizing a language for institutions and norms', in *Intelligent agents VIII*, 348–366, Springer, (2002).

[17] Yliès Falcone, Laurent Mounier, Jean-Claude Fernandez, and Jean-Luc Richier, 'Runtime enforcement monitors: composition, synthesis, and enforcement abilities', *Formal Methods in System Design*, **38**(3), 223–262, (2011).

[18] D. Gaertner, A. Garc'ia-camino, P. Noriega, and W. Vasconcelos, 'Distributed norm management in regulated multi-agent systems', in *AAMAS'07: Proceedings of the 6th International Conference on Autonomous Agents and Multiagent Systems*, pp. 624–631, (2007).

[19] Richard Gay, Heiko Mantel, and Barbara Sprick, 'Service automata', in *Formal Aspects of Security and Trust*, 148–163, Springer, (2012).

[20] Leslie Lamport, 'Proving the correctness of multiprocess programs', *Software Engineering, IEEE Transactions on*, (2), 125–143, (1977).

[21] Jay Ligatti, Lujo Bauer, and David Walker, 'Edit automata: Enforcement mechanisms for run-time security policies', *International Journal of Information Security*, **4**(1-2), 2–16, (2005).

[22] Naftaly H Minsky and Victoria Ungureanu, 'Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems', *ACM Transactions on Software Engineering and Methodology (TOSEM)*, **9**(3), 273–305, (2000).

[23] F.Y. Okuyama, R.H. Bordini, and A.C. da Rocha Costa, 'A distributed normative infrastructure for situated multi-agent organisations', in *AAMAS'08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 3*, (2008).

[24] Fred B Schneider, 'Enforceable security policies', *ACM Transactions on Information and System Security (TISSEC)*, **3**(1), 30–50, (2000).

[25] Yoav Shoham and Moshe Tennenholtz, 'On social laws for artificial agent societies: off-line design', *Artificial Intelligence*, **73**(1-2), 231–252, (1995).

[26] Maurice H Ter Beek, Clarence A Ellis, Jetty Kleijn, and Grzegorz Rozenberg, 'Synchronizations in team automata for groupware systems', *Computer Supported Cooperative Work (CSCW)*, **12**(1), 21–69, (2003).

[27] Maurice H ter Beek, Gabriele Lenzini, and Marinella Petrocchi, 'Team automata for security:–a survey–', *Electronic Notes in Theoretical Computer Science*, **128**(5), 105–119, (2005).

[28] B. Testerink and M. Dastani, 'A norm language for distributed organizations', in *BNAIC'12:Proceedings of the 24th Belgium-Netherlands Artificial Intelligence Conference*, pp. 234–241, (2012).

[29] Wamberto W Vasconcelos, Martin J Kollingbaum, and Timothy J Norman, 'Normative conflict resolution in multi-agent systems', *Autonomous Agents and Multi-Agent Systems*, **19**(2), 124–152, (2009).

[30] Zhenrong Yang, Aiman Hanna, and Mourad Debbabi, 'Team edit automata for testing security property', in *Information Assurance and Security, 2007. IAS 2007. Third International Symposium on*, pp. 235–240. IEEE, (2007).