Norm Monitoring under Partial Action Observability
Natalia Criado and Jose M. Such, Member, IEEE

Abstract—In the context of using norms for controlling multi-agent systems, a vitally important question that has not yet been addressed in the literature is the development of mechanisms for monitoring norm compliance under partial action observability. This paper proposes the reconstruction of unobserved actions to tackle this problem. In particular, we formalise the problem of reconstructing unobserved actions, and propose an information model and algorithms for monitoring norms under partial action observability using two different processes for reconstructing unobserved actions. Our evaluation shows that reconstructing unobserved actions increases significantly the number of norm violations and fulfilments detected.

Index Terms—Norm Monitoring, Action Observability.

I. INTRODUCTION

Within the Multi-agent System (MAS) area, norms are understood as means to coordinate and regulate the activity of autonomous agents interacting in a given social context [22]. The existence of autonomous agents that are capable of violating norms entails the development of control mechanisms that implement norms in agent societies.

In the existing literature, several authors have proposed infrastructures to observe agent actions and detect norm violations upon them [2], [8], [11], [12], [19], [23], [26]. The majority of these proposals have focused on providing efficient and scalable methods to monitor norms in dynamic agent societies, but they assume that all actions of agents are observable. However, this assumption is too strong because it is not necessarily true that all actions to be controlled can always be observed. One reason for this is that observing actions usually entails high costs. For example, the costs of setting, maintaining, and managing traffic radars to detect car speeds are very high, so traffic authorities usually decide to install a few of them in specific and critical locations. Another reason is that illegal actions may take place outside the institution controlled by the monitor; however, the effects of these actions can still be detected within the institution. For example, black market transactions cannot be directly observed by legal authorities, yet the corresponding money laundering transactions can be detected and sanctioned by these authorities.

Very recent work on norm monitoring under partial action observability proposes solutions to ensure complete action observability by increasing the actions that are observed, either by adding more monitors [7] or by adapting the norms to what can be observed [3]. However, these solutions are not always appropriate or feasible. For instance, in e-markets, such as eBay or Amazon, it is not possible to change trading laws to what can be observed. This paper goes beyond these approaches by also considering actions that were not observed but that can be reconstructed from what was observed.

The main contributions of this paper are: (i) a formalisation of the problem of reconstructing unobserved actions from observed actions for the purpose of norm monitoring; (ii) an exhaustive and an approximation solution to this problem; and (iii) an information model and algorithms used to monitor norms under partial action observability. Through an extensive empirical evaluation, we show that reconstructing unobserved actions increases noticeably the number of norm violations and fulfilments detected.

This paper is organised as follows: Section II contains the preliminary definitions used in this paper. Section III describes the information model of norm monitor proposed in this paper. Section IV contains the algorithms executed by norm monitors. Our proposal is evaluated in Section V. Related work is discussed in Section VI. Finally, conclusions are contained in Section VII.

II. PRELIMINARY DEFINITIONS

\( \mathcal{L} \) is a first-order language containing a finite set of predicate and constant symbols, the logical connective \( \neg \), the equality (inequality) symbol \( = \) (\( \neq \)), the true \( \top \) and false propositions \( \bot \), and an infinite set of variables. The predicate and constant symbols are written as any sequence of alphanumeric characters beginning with a lower case letter. Variables are written as any sequence of alphanumeric characters beginning with a capital letter. We will relate our formulae via logical entailment \( \vdash (\gamma) \). We also assume the standard notion of substitution of variables; i.e., a substitution \( \sigma \) is a finite and possibly empty set of pairs \( Y/y \) where \( Y \) is a variable and \( y \) is a term.

The set of grounded atomic formulas of \( \mathcal{L} \) is built of a finite set of predicates and objects that characterise the properties of the world relevant to norm monitoring. By a situation, we mean the properties that are true at a particular moment. Some of these properties are static and not altered by action execution, whereas other properties are dynamic and changed due to agent actions. Specifically, we represent static properties as a set of atomic grounded formulas of \( \mathcal{L} \), denoted by \( g \). A state \( s \) is a set of grounded atomic formulas of \( \mathcal{L} \), describing dynamic properties which hold on state \( s \). Thus, a situation is built on a “closed assumption” and defined by a set of static properties \( g \) and a state \( s \). Moreover, there is a set of inference rules (\( \triangledown \)) representing domain knowledge.

**EXAMPLE 1.** In this paper we will use a running example in which there are three robots that should attend requests at

\[ \text{http://www.ebay.com} \]
\[ \text{http://www.amazon.com} \]

*In this paper sets are to be interpreted as the conjunction of their elements.*
six offices in a building. The goal of the robots is to attend these requests as soon as possible. Figure [a] depicts our initial scenario. In our example, the language \( L \) contains: 4 predicate symbols (\( \text{robot}, \text{office}, \text{in}, \text{corridor} \)), used to represent the robots and offices, the positions of the robots and the connections between offices in the building; 3 constant symbols to represent the robots (\( r_1, r_2, r_3 \)); and 6 constant symbols to represent the offices (\( a, b, c, d, e, f \)). The information about the robots, offices and corridors between offices is static and represented as follows:

\[
g = \{ \text{robot}(r_1), \text{robot}(r_2), \text{robot}(r_3), \text{office}(a), \ldots, \text{office}(f), \text{corridor}(a,b), \text{corridor}(b,a), \ldots, \text{corridor}(e,a) \}
\]

The information about the location of the robots is dynamic. Specifically, the initial state \( s_0 \) is defined as follows:

\[
s_0 = \{ \text{in}(r_1,a), \text{in}(r_2,d), \text{in}(r_3,e) \}
\]

In this domain there is an inference rule (\( \triangledown \)) representing that a robot cannot be in two different offices at the same time:

\[
\triangledown = \{ \{ \text{in}(R_1,OA), \text{in}(R_1,OB), OA \neq OB \} \cup \bot \}
\]

A. Action Definitions

\( D \) is a finite set of action descriptions that induce state transitions. An action description \( d \) is represented using preconditions and postconditions. If a situation does not satisfy the preconditions, then the action cannot be applied in this situation. In contrast, if the preconditions are satisfied, then the action can be applied transforming the current state into a new state in which all negative literals appearing in the postconditions are deleted and all positive literals in the postconditions are added. Moreover, actions are executed in a MAS and, as a consequence, we need to be able to represent concurrent actions with interacting effects. For the sake of simplicity, we will represent concurrent actions without an explicit representation of time\(^4\) as proposed in \[6\]. The main idea behind this representation is that individual agent actions do interact (i.e., one action might only achieve the intended effect if another action is executed concurrently). Specifically, each action is also represented by a (possibly empty) concurrent condition that describes the actions that must (or cannot) be executed concurrently.

**Definition 1.** An action description \( d \) is a tuple \(<\text{name}, \text{pre}, \text{con}, \text{post}>\) where:

- \text{name} is the action name;
- \text{pre} is the precondition, i.e., a set of positive and negative literals of \( L \) (containing both dynamic and static properties) as well as equality and inequality constraints on the variables;
- \text{con} is the concurrent condition; i.e., a set of positive and negative action schema\(^3\) some of which can be partially instantiated or constrained;
- \text{post} is the postcondition; i.e., a set of positive and negative literals of \( L \) (containing dynamic properties only).

Given an action description \( d \), we denote by \( \text{pre}(d) \), \( \text{con}(d) \), \( \text{post}(d) \) the action precondition, concurrent condition and postcondition.

**Example 2.** In our example, there is only one action that can be executed by robots:

\[
\langle \text{move}, \{ \text{robot}(R), \text{office}(O1), \text{office}(O2), \text{in}(R,O1), \text{in}(R,O2) \} \rangle
\]

This action represents the movement of a robot from one office to another. The parameters of this action are the robot \( R \), the source office \( O1 \), the destination office \( O2 \). To execute this action, the robot should be located at the source office and the two offices should be connected. Once the operation has been applied, the robot is no longer at the source office and it is at the destination office.

Finally, note that it is possible to define a \( \text{NOP} \) action that allows agents to remain still. For the sake of clarity and simplicity, the running example does not include the \( \text{NOP} \) action. However, our proposal is agnostic with respect to the existence of a \( \text{NOP} \) action and we have carried out experiments that include the \( \text{NOP} \) action (see Section [V-B]).

**Definition 2.** Given a situation represented by the state \( s \) and a set of static properties \( g \), and an action description \( d = <\text{name}, \text{pre}, \text{con}, \text{post}>; \) an action instance (or action) is a tuple \(<\text{name}, \text{pre}', \text{con}', \text{post}'>\) such that:

- There is a substitution \( \sigma \) of variables in \( \text{pre} \), such that the precondition is satisfied (i.e., entailed by) the situation; i.e., \( s, g \vdash \sigma \cdot \text{pre} \);
- \( \sigma \cdot \text{pre}, \sigma \cdot \text{post} \) are grounded;
- \( \text{pre}' \) is a set of grounded literals in \( \sigma \cdot \text{pre} \) containing dynamic properties only;
- \( \text{post}' = \sigma \cdot \text{post} \) and \( \text{con}' = \sigma \cdot \text{con} \).

\(^3\)A more sophisticated definition of the concurrent condition would allow actions to have conditional effects according to the actions that are executed concurrently. Without loss of expressiveness, we will not consider conditional effects in action descriptions (note that any action with conditional effects can be represented by a set of actions with non conditional effects).

\(^4\)An action schema contains an action name and the parameters of this action. Note that positive action schemata are implicitly existentially quantified –i.e., one instance of each positive schema must occur concurrently– and negative schemata are implicitly universally quantified.
Given an action \( a \), we denote by \( \text{actor}(a) \) the agent performing the action, and by \( \text{pre}(a), \text{con}(a), \text{post}(a) \) the precondition, concurrent condition and postcondition.

**Example 3.** In state \( s_0 \), the robot \( r_1 \) moves from office \( a \) to office \( b \). This is formalised as follows:

\[
\left\{ \text{move}, \{ \text{robot}(r_1), \text{office}(a), \text{office}(b), \text{in}(r_1, a), \text{corridor}(a,b), \{\}, \{\neg \text{in}(r_1,a), \text{in}(r_1,b)\} \right\}
\]

In a MAS, concurrent actions\(^7\) define state transitions. More formally, a concurrent action \( A = \{a_1, ..., a_n\} \) is a set of individual actions. Given a set of actions \( A = \{a_1, ..., a_n\} \), we define \( \text{pre}(A) = \bigcup \text{pre}(a_i) \), \( \text{post}(A) = \bigcup \text{post}(a_i) \) and \( \text{actor}(A) = \bigcup \text{actor}(a_i) \).

Given a concurrent action \( A = \{a_1, ..., a_n\} \) we say that the concurrent condition of an individual action \( a_i \) of \( A \) is satisfied when for all positive schema in the concurrent condition exists an action \( a_j \) (\( i \neq j \)) in \( A \), such that \( a_j \) is an instance of the schema; and for all negative schema none of the elements in \( A \) is an instance of the schema. For the sake of simplicity, we assume that each agent performs one action at a time\(^8\).

**Definition 3.** (Consistency [6]) Given a concurrent action \( A = \{a_1, ..., a_n\} \) it is consistent if:
- \( \text{pre}(A) \) is consistent (i.e., \( \text{pre}(A) \not\models \bot \));
- \( \text{post}(A) \) is consistent (i.e., \( \text{post}(A) \not\models \bot \));
- the concurrent condition of each action is satisfied;
- the concurrent action is complete; i.e., each agent performs one action (possibly the NOP action) in \( A \).

**Example 4.** The concurrent action \( A = \{\text{move}(r_1,a,b), \text{move}(r_2,d,a), \text{move}(r_3,e,a)\} \) is consistent since:
- \( \text{pre}(A) = \{\text{in}(r_1,a), \text{in}(r_2,d), \text{in}(r_3,e)\} \) which is consistent;
- \( \text{post}(A) = \{\text{in}(r_1,b), \text{in}(r_2,a), \text{in}(r_3,a), \neg \text{in}(r_1,a), \neg \text{in}(r_2,d), \neg \text{in}(r_3,e)\} \) which is consistent;
- the concurrent conditions of both actions are satisfied;
- each robot performs one action.

A concurrent action \( A = \{a_1, ..., a_n\} \) is applicable in a situation if \( A \) is consistent\(^9\) and each individual action \( a_i \in A \) is applicable in this situation.

Given a consistent action, we define its effects as the postconditions of its individual actions and the preconditions not invalidated by the postconditions. More formally, given a concurrent action \( A = \{a_1, ..., a_n\} \) its effects are a set of grounded literals as follows:

\[
\text{eff}(A) = \left( \bigcup_{\text{pre}\in\text{pre}(A)} \text{pre}\right) \bigcup \left( \bigcup_{\text{post}\in\text{post}(A)} \text{post} \right)
\]

\(^7\)Concurrent action means actions that occur at the same time and does not necessarily imply agent cooperation or coordination.

\(^8\)This limitation can be relaxed by decomposing agents into groups of agents corresponding to agents’ actuators [6].

\(^9\)For simplicity, we represent actions by their schemata.

\(^{10}\)Note that we assume that actions executed by agents at a time are consistent, which entails that observed actions are consistent. Relaxing this assumption may not be desirable, as this would entail issues such as that mutually exclusive actions can be executed simultaneously. For example, the world cannot be in an inconsistent situation in which two robots have moved a piece simultaneously and the piece is in two different locations at the same time.

### B. Norm Definitions

We consider norms as formal statements that define patterns of behaviour by means of deontic modalities (i.e., obligations and prohibitions). Specifically, our proposal is based on the notion of norm as a conditional rule of behaviour that defines under which circumstances a pattern of behaviour becomes relevant and must be fulfilled [23], [22].

**Definition 4.** A norm is defined as a tuple \( \langle \text{deontic, condition, action}\rangle \), where:
- \( \text{deontic} \in \{\text{O, P}\} \) is the deontic modality of the norm, determining if the norm is an obligation (O) or prohibition (P);
- \( \text{condition} \) is a set of literals of \( L \) as well as equality and inequality constraints that represents the norm condition, i.e., it denotes the situations in which the norm is relevant;
- \( \text{action} \) is a positive action schema that represents the action controlled by the norm.

**Example 5.** In our example, there is a norm that avoids collisions by forbidding any robot to move into an office when the office is occupied by another robot:

\( \langle \text{P, in}(R_1, L_2), \text{move}(R_2, L_1, L_2)\rangle \)

This norm states that when a robot \( R_1 \) is located in office \( O_2 \) other robots are forbidden to move from any office \( L_1 \) to \( L_2 \).

In line with related literature [2], [3], we consider a closed legal system, where everything is considered permitted by default, and obligation and prohibition norms define exceptions to this default permission rule. We also define that a norm is relevant to a specific situation if the norm condition is satisfied in the situation. Besides, we define that a norm condition is satisfied in a given situation when there is a substitution of the variables in the norm condition such that the constraints in the norm condition are satisfied and the positive (vs. negative) literals in the norm condition are true (vs. false) in the situation.

**Definition 5.** Given a specific situation denoted by a state \( s \) and a set of static properties \( g \), and a norm \( \langle \text{deontic, condition, action}\rangle \); a norm instance is a tuple \( \langle \text{deontic, action'}\rangle \) such as:

- There is a substitution \( \sigma \) such that the condition is satisfied in the situation; i.e., \( s,g \vdash \sigma \cdot \text{condition} \);
- \( \text{action'} = \sigma \cdot \text{action} \).

**Example 6.** In state \( s_0 \) the norm that forbids robots to move into occupied offices is instantiated as follows:

\( \langle \text{P, move}(R_2, L_1, d) \rangle \) where \( \sigma = \{L_2/d\} \)

\( \langle \text{P, move}(R_2, L_1, a) \rangle \) where \( \sigma = \{L_2/a\} \)

\( \langle \text{P, move}(R_2, L_1, e) \rangle \) where \( \sigma = \{L_2/e\} \)

The semantics of instances (and norms in general) depends on their deontic modality. An obligation instance is fulfilled when the mandatory action is performed and violated otherwise, while a prohibition instance is violated when the forbidden action is performed and fulfilled otherwise. We
classify detected violations (vs. fulfilments) into: identified violation (vs. fulfilment), which refers to when the monitor knows the specific action that an agent has executed and violates (vs. fulfills) an instance; and discovered violation (vs. fulfilment), which refers to when the monitor knows that an agent has violated (vs. fulfilled) some instance but does not know the forbidden (vs. mandatory) action executed by the agent.

III. NM INFORMATION MODEL

Let us assume a set of agents \( Ag \) to be monitored, a set of norms \( N \) that regulate the actions of agents, and a set \( D \) of action descriptions that represent the actions that can be performed by agents. For the sake of simplicity, we assume that there is a single Norm Monitor (NM) that observes the actions performed by agents and monitors norm compliance. We also assume that actions are deterministic and that the current state evolves due to action execution only. The goal of the NM is to analyse a partial sequence of action observations to detect norm violations. The enforcement of norms is out of the scope of this work and we assume that once the NM detects a norm violation (vs. fulfilment), it applies the corresponding sanction (vs. reward).

A. State Representation

As the NM only observes a subset of the actions performed by agents, it has partial information about the state of the world. The NM represents each partial state of the world, denoted by \( p \), using an “open world assumption” as a set of grounded literals that are known in the state. Thus, a partial state contains positive (vs. negative) grounded literals representing dynamic properties known to be true (vs. false) in the state. The rest of dynamic properties are unknown.

The NM has some surveillance cameras to monitor the movement of robots in the building. Specifically, the corridors that are monitored are the ones between offices: a and b; b and c; and b and f. These corridors are represented by black arrows in Figure 1a whereas non-monitored corridors are represented by grey arrows. In the initial state \( s_0 \) depicted in Figure 1a, the robots execute the actions \( move(r_1, a, b) \) and \( move(r_3, c, a) \) resulting in a new state \( s_1 \) depicted in Figure 1b. However, the NM only observes the action of robot \( r_1 \), because this action takes place in a monitored corridor; i.e., \( Act_0 = \{ move(r_1, a, b) \} \). In the next state \( s_1 \), the robots execute actions \( move(r_1, b, c) \), \( move(r_2, a, e) \) and \( move(r_3, a, b) \) resulting in a new state \( s_2 \) depicted in Figure 1c. In this case the NM observes two actions; i.e., \( Act_1 = \{ move(r_1, b, c), move(r_3, a, b) \} \). Considering these two sets of observed actions, the NM is able to infer the dynamic propositions that are known in \( s_1 \) as follows:

\[
p_1 = \{ in(r_1, b), \neg in(r_1, a), in(r_3, a) \}
\]

If the NM uses the information about the states and the observed actions, then no violation of the norm is detected and

\[\text{Definition 6. Given a partial state description } p_t \text{ corresponding to time } t, \text{ and two consecutive sequences of observed actions } Act_t \text{ and } Act_{t+1} \text{ executed by agents at times } t \text{ and } t + 1, \text{ respectively; the new partial state } p_{t+1} \text{ resulting from executing actions } Act_t \text{ in } p_t \text{ and actions } Act_{t+1} \text{ in } p_{t+1} \text{ is obtained as follows:}
\]

\[
p_{t+1} = \begin{cases} 
\text{post}(Act_t) \bigcup \text{pre}(Act_{t+1}) & \text{if } |Act_t| < |Ag| \\
p^*_t \bigcup \text{eff}(Act_t) \bigcup \text{pre}(Act_{t+1}) & \text{otherwise}
\end{cases}
\]

where \( p^*_t \) is the set of invariant literals; i.e., literals of \( p_t \) that have not been modified by the actions in \( Act_t \) and it is defined as follows:

\[
p^*_t = \bigcup_{l \in \text{eff}(Act_t) \forall l} t
\]

Example 7. In our example, the NM knows which grounded atomic formulas are true or false in the initial state:

\[
p_0 = \{ \text{in}(r_1, a), \neg \text{in}(r_1, b), \neg \text{in}(r_1, c), \neg \text{in}(r_1, d), \neg \text{in}(r_1, e), \neg \text{in}(r_1, f), \text{in}(r_2, d), \neg \text{in}(r_2, a), \neg \text{in}(r_2, b), \neg \text{in}(r_2, c), \neg \text{in}(r_2, e), \neg \text{in}(r_2, f), \text{in}(r_3, c), \neg \text{in}(r_3, d), \neg \text{in}(r_3, f) \}
\]

However, the NM only observes the action of robot \( r_1 \), because this action takes place in a monitored corridor; i.e., \( Act_0 = \{ \text{move}(r_1, a, b) \} \). In the next state \( s_1 \), the robots execute actions \( \text{move}(r_1, b, c) \), \( \text{move}(r_2, a, e) \) and \( \text{move}(r_3, a, b) \) resulting in a new state \( s_2 \) depicted in Figure 1c. In this case the NM observes two actions; i.e., \( Act_1 = \{ \text{move}(r_1, b, c), \text{move}(r_3, a, b) \} \). Considering these two sets of observed actions, the NM is able to infer the dynamic propositions that are known in \( s_1 \) as follows:

\[
p_1 = \{ \text{in}(r_1, b), \neg \text{in}(r_1, a), \text{in}(r_3, a) \}
\]

\[\text{Inference rules might also be used together with observed actions to infer more propositions. However, we propose to conduct this inference process once unobserved actions have been reconstructed to increase the number of propositions that are inferred, as described in Section III-B.} \]
B. Action Reconstruction

NMs use Definition 6 to generate partial state descriptions based on the observed actions. Additionally, we propose that NMs reconstruct the actions that have not been observed. This reconstruction process entails: (i) searching for the actions that have been performed by unobserved agents; and (ii) using the actions found to increase the knowledge about the state of the world. The reconstruction process must be sound, e.g., it cannot indicate that a violation has occurred when it has not in fact occurred. In the following, we introduce full and approximate methods for reconstructing unobserved actions.

1) Full Reconstruction: Full reconstruction tries to find exhaustively the actions performed by all the agents that have not been observed. To this aim, the full reconstruction performs a search to identify all solutions to the reconstruction problem.

Definition 7. Given a partial state description $p_i$ corresponding to time $t$ (named initial state), a set of observed actions $Act_t$ at time $t$, and an partial resulting state $p_{i+1}$ corresponding to time $t + 1$ (named final state); we define search as a function that computes sets of solutions $S = \{S_1, ..., S_k\}$ such that each solution $S_i$ in $S$ is a set of actions such that:

- the concurrent action $S_i \cup Act_t$ is consistent;
- the initial state induced by the concurrent action $S_i \cup Act_t$ is consistent (i.e., $g, p_i, \text{pre}(S_i \cup Act_t), \forall \parallel \bot$);
- the final state induced by the concurrent action $S_i \cup Act_t$ is consistent (i.e., $g, p_{i+1}, \text{post}(S_i \cup Act_t), \forall \parallel \bot$).

Thus, a solution is a set of actions performed by the agents that have not been observed that are consistent with the states of the world before and after the execution of the actions. Given that the NM has a partial knowledge of the states, we do not require that the preconditions (vs. postconditions) of actions in a solution are met in the initial (vs. final) state, since it is possible that the preconditions (vs. postconditions) are true, but the NM is unaware of it.

EXAMPLE 8. Given the partial state description $p_0$ (initial state), the set of observed actions $Act_0$, and the partial resulting state $p_1$ (final state), the search function looks for actions of agents $r2$ and $r3$ (since they are the agents that have not been observed). According to the initial position of $r2$, the NM can infer that $r2$ may have performed two different actions move($r2, d, a$) and move($r2, d, e$) — these two actions are the only ones consistent with $p_0$. Similarly, the NM can infer that $r3$ may have performed three different actions move($r3, e, a$), move($r3, e, d$) and move($r3, e, f$) — these three actions are the only ones consistent with $p_0$. However, the actions move($r3, e, d$) and move($r3, e, f$) are not consistent with the final state — recall that these two actions have as postcondition the fact that $r3$ is in offices $d$ and $f$, respectively; that $p_1$ states that $r3$ is in office $a$ (i.e., $p_1$ was defined as $\{\text{in}(r1, b), \neg\text{in}(r1, a), \text{in}(r3, a)\}$ in Example 2); and that $\forall$ defines as inconsistent states where any robot is at two different locations. As a result, the solution set for this problem is defined as:

$$S = \{\{\text{move}(r2, d, a), \text{move}(r3, e, a)\}, \{\text{move}(r2, d, e), \text{move}(r3, e, a)\}\}$$

Once all solutions are found, the NM uses this information to extend the information about the actions performed by unobserved agents and the state of the world. To ensure that the reconstruction is sound, the NM calculates the intersection of actions in the solutions to select actions it is completely sure about (i.e., actions belonging to all solutions). Given a set of search solutions $S = \{S_1, ..., S_k\}$ for some initial and final states, we define the reconstruction action set as follows:

$$R = \bigcap_{S_i \in S} S_i$$

If $R \neq \emptyset$, then the NM expands its knowledge about the actions performed by agents and it uses this information to increase the knowledge about the initial and final states. More formally, the set of actions observed in $t$ is updated as:

$$Act_t = Act_t \cup R$$

The initial state $p_t$ is updated as follows:

$$p_t = p_t \bigcup \text{pre}(R)$$

Finally, the final state is updated as follows:

$$p_{t+1} = \begin{cases} p_{t+1} \bigcup \text{post}(R) \bigcup p^*_t & \text{if } |Act_t| < |Ag| \\ p_{t+1} \bigcup p^*_t \bigcup \text{eff}(Act_t) & \text{otherwise} \end{cases}$$

where $p^*_t$ is defined as before and $p^*_t$ is the set of extended invariant literals; i.e., literals in $p_t$ that have not been modified since there is not a solution $S_i \in S$ that the concurrent action $S_i \cup Act_t$ changes any of these literals:

$$p^*_t = \bigcup_{S_i \in S} \bigcup_{l \in p_t} \forall \parallel \bot$$

EXAMPLE 9. The reconstruction set for the example is:

$$R = \{\text{move}(r3, e, a)\}$$

This action belongs to all solutions, so the NM can be absolutely sure about the performance of this action, even when the NM has not observed it. As a consequence, the NM extends its information as follows:

$$Act_0 = \{\text{move}(r1, a, b), \text{move}(r3, e, a)\}$$

and $p_0$ remains unchanged and $p_1$ is updated as follows:

$$p_1 = \{\text{in}(r1, b), \neg\text{in}(r1, a), \neg\text{in}(r1, c), \neg\text{in}(r1, d), \neg\text{in}(r1, e), \neg\text{in}(r1, f), \text{in}(r2, b), \text{in}(r2, c), \text{in}(r2, f), \text{in}(r3, a), \neg\text{in}(r3, b), \neg\text{in}(r3, c), \neg\text{in}(r3, d), \neg\text{in}(r3, e), \neg\text{in}(r3, f)\}$$

\footnotetext[14]{Note that proposition $\neg\text{in}(r2, d)$ cannot be inferred since our mechanism can also be used in scenarios with the NOP action and, as a result, it cannot assume that agents must move in each step.}

\footnotetext[15]{If all actions were observed, no reconstruction would be needed.
The main disadvantage of full reconstruction is that, for many real-world problems, the number of candidate solutions that needs to be explored is prohibitively large, as shown later in Section IV. In response to this problem, we provide a polynomial approximation below.

2) Approximate Reconstruction: Approximate reconstruction includes an approximate search that finds the actions performed by unobserved agents that are consistent with the states of the world before and after action execution. Specifically, approximate reconstruction identifies actions that do not necessarily include the specific actions performed by unobserved agents but that allow the NM to control norms. The main intuition behind approximate reconstruction is as follows: imagine that at a given initial state an agent can perform just one action and that this action is forbidden (vs. mandatory). In this case, the NM identifies that the agent has violated (vs. fulfilled) a norm. Besides that, if an agent can perform n different actions and all these actions are forbidden (vs. mandatory), the NM does not need to know which action has been executed to conclude that a norm has been violated (vs. fulfilled). Hence, we say that a violation has been discovered (instead of identified). Given a set of prohibition instances P and an action a, we define that the action a is forbidden (denoted by forbidden(P, a)) when ∃p ∈ P : ∃σ : σ ∗ action(p) = a. Similarly, given a set of obligation instances O and an action a, we define that the action a is mandatory (denoted by mandatory(O, a)) when ∃o ∈ O : ∃σ : σ ∗ action(o) = a.

Definition 8. Given a partial state pᵣ, a set of observed actions Actᵣ, and a partial resulting state pᵣ₊₁; we define approximate search as a function that calculates the set of all unobserved applicable actions S = {a₁, ..., aₙ} such that:
- the preconditions of each action in S are consistent with the initial state (i.e., ∀aᵢ ∈ S : g, pᵣ, pre(aᵢ), ∨ ⊥ ⊥);
- the postconditions of each action in S are consistent with the final state (i.e., ∀aᵢ ∈ S : g, pᵣ₊₁, post(aᵢ), ∨ ∨ ⊥);
- actions in S are performed by unobserved agents (i.e., actor(S) ∩ actor(Actᵣ) = ∅);
- all unobserved agents perform at least one action in S.

Example 10. Given the partial state description p₀ (initial state), the set of observed actions Act₀, and the partial resulting state p₁ (final state), the approximate search function looks for actions of agents r₂ and r₃ (since they are the agents that have not been observed). According to the initial position of r₂, the NM can infer that r₂ may have performed two different actions move(r₂, d, a) and move(r₂, d, e). Again, r₃ may have performed action move(r₃, e, a) —recall that actions move(r₃, e, d) and move(r₃, e, f) are inconsistent with the final state p₁ since it was defined as \{in(r₁, b), ¬in(r₁, a), in(r₃, a)\} in Example 2.

The approximate solution for this problem is defined as:
\[ S = \{move(r₂, d, a), move(r₂, d, e), move(r₃, e, a)\} \]

As in full reconstruction, the NM uses approximate search solutions (S) to expand its knowledge about the actions performed by unobserved agents and to increase the knowledge about the initial and final states. When an unobserved agent may have executed only one action, then the NM knows for sure that this action was executed. More formally, the reconstruction action set is defined as follows:
\[ R = \bigcup_{a \in S : \exists a' \in S} a \]

The set of actions observed in t is updated as:
\[ Actᵣ = Actᵣ \cup R \]

Then the initial state pᵣ is updated as follows:
\[ pᵣ = pᵣ \cup pre(R) \]

The final state is updated as follows:
\[ pᵣ₊₁ = \begin{cases} pᵣ₊₁ \cup \text{post}(R) \cup pᵣ^⋆ & \text{if } |Actᵣ| < |Ag| \\ pᵣ₊₁ \cup pᵣ^⋆ \cup \text{eff}(Actᵣ) & \text{otherwise} \end{cases} \]

where \( pᵣ^⋆ \) is defined as before and \( pᵣ^⋆ \) is the set of extended invariant literals in \( pᵣ \); i.e., literals that have not been modified since there is not an observed action or an applicable action that changes them.

Finally, the set of discovered violations and fulfillments is a set of actions:
\[ D = \{a₁, ..., a₃\} \]

where for each action \( a \) in \( D \): a is in \( S \) and the set of unobserved applicable actions for the agent that performs \( a \) (i.e., \{\( a' : a' \in S \land \text{actor}(a') = \text{actor}(a)\}) is formed by two or more forbidden (vs. mandatory) actions; and a is one of these forbidden (vs. mandatory) actions.

When an agent is only able to perform forbidden (vs. mandatory) actions, an action among these can be selected according to various criteria. For example, in a normative system where the presumption of innocence principle holds, the NM should assume that the agent has violated (vs. fulfilled) the least (vs. most) important norm and the action that violates (vs. fulfills) this norm is selected. Note that discovering violations is very useful in many practical applications, in which it would allow the NM to ban offender agents (e.g., Intrusion Detection/Prevention Systems [4]), to stop the execution of any offender agent (e.g., Business Process Compliance monitoring [29]), or to put offender agents under close surveillance (e.g.,
Model-Based Diagnosis Systems (24), even when the specific action performed is not known.

**Example 11.** In case of the approximate reconstruction, r3 is only able to perform one action, which entails that the NM can be sure about the performance of this action and the reconstruction set is defined as:

$$R = \{move(r3, e, a)\}$$

As a consequence, the NM extends its information as follows:

$$Act_0 = \{move(r1, a, b), move(r3, e, a)\}$$

p0 remains unchanged and p1 is updated as follows:

$$p_1 = \{in(r1, b), \neg in(r1, a), \neg in(r1, c), \neg in(r1, d), \neg in(r1, e), \neg in(r2, b), \neg in(r2, c), \neg in(r2, f), in(r3, a),$$

$$\neg in(r3, b), \neg in(r3, c), \neg in(r3, d), \neg in(r3, e), \neg in(r3, f)\}$$

In this situation, r2 is only able to execute forbidden actions —recall that the instances \(\langle P, move(R2, L1, a)\rangle\) and \(\langle P, move(R2, L1, e)\rangle\) forbid any robot to move into offices a and e and that r2 may have executed actions \(move(r2, d, a)\) and \(move(r2, d, e)\). Thus, the set of discovered violations and fulfilments can be defined as either containing \(move(r2, d, a)\) or \(move(r2, d, e)\). Note that the discovered violation does not correspond to the action executed by r2, however, it allows the NM to determine that r2 must have violated an instance.

### C. Norm Monitoring

Once all the information about the actions performed by the agents and the partial states have been reconstructed, the NM checks the actions of agents to determine which instances have been violated or fulfilled. Recall that norms in our model are defined as conditional rules that state which actions are obligatory or forbidden. Given that the NM has partial knowledge about the state of the world, the NM should control norms only when it is completely sure that the norms are relevant to ensure that the norm monitoring process is sound. In particular, we define that a norm is relevant to a partial situation when the norm condition is satisfied by the partial situation —i.e., a norm (deontic, condition, action) is relevant to a partial situation represented by a partial state p, the static properties g and the domain knowledge v if \(\exists \sigma\) such that \(p, g, v \models \sigma \cdot condition\).

**Example 12.** In state p0 the norm that forbids robots to move into occupied offices is instantiated three times as follows:

\[
\begin{align*}
\langle P, move(R2, L1, d) \rangle & \text{ where } \sigma = \{L2/d\} \\
\langle P, move(R2, L1, a) \rangle & \text{ where } \sigma = \{L2/a\} \\
\langle P, move(R2, L1, e) \rangle & \text{ where } \sigma = \{L2/e\}
\end{align*}
\]

Once the NM has determined which norm instances hold in a given situation, it has to check the actions of agents to determine which instances have been violated and which ones have been fulfilled.

**Obligation Instance.** In presence of partial knowledge about the actions performed by agents, the NM can only determine that an obligation instance has been fulfilled. If the NM knows all the actions performed by agents, then it can determine whether an obligation has been fulfilled or violated.

**Definition 9.** Given an obligation instance \(\langle O, action'\rangle\) and a set of observed actions \(Act\), then the obligation is defined as:

\[
\begin{align*}
\text{fulfilled} & \iff \exists \sigma : \sigma \cdot action' \in Act \\
\text{violated} & \iff (\not\exists \sigma : \sigma \cdot action' \in Act) \land |Act| = |Ag| \\
\text{unknown} & \text{otherwise}
\end{align*}
\]

**Prohibition Instance.** In presence of partial knowledge about the actions performed by agents, the NM can only determine that a prohibition instance has been violated. If the NM knows all the actions performed by agents then it can determine whether a prohibition has been fulfilled or violated.

**Definition 10.** Given a prohibition instance \(\langle P, action'\rangle\) and a set of observed actions \(Act\), then the prohibition is defined as:

\[
\begin{align*}
\text{violated} & \iff \exists \sigma : \sigma \cdot action' \in Act \\
\text{fulfilled} & \iff (\not\exists \sigma : \sigma \cdot action' \in Act) \land |Act| = |Ag| \\
\text{unknown} & \text{otherwise}
\end{align*}
\]

Finally, the set of discovered violations and fulfilments is used to identify those agents that have violated or fulfilled an instance.

**Example 13.** Taking into account the set of actions \(Act_0\), the NM can identify that robot r3 has violated the instance \(\langle P, move(R2, L1, a)\rangle\), even though this forbidden action has not been observed by the NM. Specifically, there is \(\sigma = \{R2/r3, L1/e\}\) such that \(\sigma(move(R2, L1, a)) \in Act_0\). Besides that, the approximate reconstruction discovers that robot r2 has violated a prohibition instance though it does not know the exact action performed —recall that either \(move(r2, d, a)\) or \(move(r2, d, e)\) are in D. Had the NM not performed the proposed reconstruction processes, none of these violations would have been detected.

### IV. NM Algorithms

Algorithm 1 contains the NM pseudocode. In each step, the NM observes the actions of agents and uses this information to update the current and the previous partial states (lines 4-9). If all the actions have not been observed in the previous state, then the NM executes the reconstruction algorithm to reconstruct unobserved actions (lines 11-14). Then, the checkNorms function is executed to determine which norms have been violated and fulfilled in the previous state (line 15) according to Definitions 9 and 10.

Note that the NM code can be executed while actions are performed without delaying agents. Regarding the temporal cost of the algorithm executed by NMs, it is determined by the cost of the reconstruction algorithm, the implementations of which (full and approximate) are discussed below.

17Note forbidden actions do not necessarily lead to a collision since neither robot r1 nor r3 have moved to office d.
Algorithm 1 NM Algorithm

Require: $Ag, N, D, \forall, g$  
1: $p_0 = \emptyset$  
2: $t \leftarrow 0$  
3: while true do  
4: $Act_t \leftarrow \text{observeActions}()$  
5: if $|Act_t| < |Ag|$ then  
6: $p_{t+1} \leftarrow \text{post}(Act_t)$  
7: else  
8: $p_{t+1} \leftarrow p_t \land \text{eff}(Act_t)$  
9: $p_t \leftarrow p_t \land \text{pre}(Act_t)$  
10: if $t > 0$ then  
11: if $|Act_{t-1}| < |Ag|$ then  
12: $TA \leftarrow Ag \setminus \text{actors}(Act_{t-1})$  
13: $D \leftarrow \emptyset$  
14: $\text{reconstruction}(p_{t-1}, p_t, Act_{t-1}, TA, D)$  
15: $\text{checkNorms}(p_{t-1}, Act_{t-1})$  
16: $t \leftarrow t + 1$

Full Reconstruction (Algorithm 2). This pseudocode corresponds to the full reconstruction algorithm. This algorithm calls the function search to search the actions of target agents (line 2). Then, for all the solutions found, the NM checks if they are consistent according to Definition 7 (lines 4-6). Finally, consistent solutions are used to extend the set of observed actions and the knowledge about the initial and final states (lines 7-14).

Algorithm 2 Full Reconstruction Algorithm

1: function FULLRECONSTRUCTION(i, f, Act, TA, D)  
2: $S' \leftarrow \text{search}(i, f, Act, TA, D)$  
3: $S \leftarrow \emptyset$  
4: for all $S_j \in S'$ do  
5: if $\text{checkSolutionConsistency}(Act, S_j, i, f)$ then  
6: $S \leftarrow S \cup S_j$  
7: $R \leftarrow \bigcap_{S \in S} S_i$  
8: if $R \neq \emptyset$ then  
9: $Act \leftarrow Act \cup R$  
10: $i \leftarrow i \land \text{pre}(R)$  
11: if $|Act| < |Ag|$ then  
12: $f \leftarrow f \land \text{post}(R) \land i^*$  
13: else  
14: $f \leftarrow i^* \land f \land \text{eff}(Act)$

Approximate Reconstruction (Algorithm 3). This algorithm calls the function ApproximateSearch to search the applicable actions per each target agent (line 2). Then, the list of applicable actions per each agent is checked (lines 3-12). Specifically, if an agent may have executed one action only, then the NM knows that this action was executed and it updates the reconstructed action set (lines 3-5). Then, the set of observed actions and the knowledge about the initial and final states is updated (lines 6-12). Finally, discovered violations and fulfilments are calculated (lines 14-19).

Algorithm 3 Search Function

1: function SEARCH(i, f, Act, TA)  
2: if $TA \neq \emptyset$ then  
3: for all $d \in D$ do  
4: if $\exists \sigma : \text{checkActionConsistency}(\sigma \cdot d, i, f) \land \text{actor}(\sigma \cdot d) \in TA$ then  
5: $\alpha = \text{actor}(\sigma \cdot d)$  
6: $i' \leftarrow i \land \text{pre}(\sigma \cdot d)$  
7: $f' \leftarrow f \land \text{post}(\sigma \cdot d)$  
8: $Act' \leftarrow Act \land \sigma \cdot d$  
9: $TA' \leftarrow TA \setminus \{\alpha\}$  
10: $S' \leftarrow \text{search}(i', f', Act', TA')$  
11: for all $S_i \in S$ do  
12: $S_i \leftarrow S_i \cup \sigma \cdot d$  
13: return $S'$

Algorithm 4 contains the pseudocode of the recursive search function that computes all the sequences of consistent actions that may have been executed by the agents that have not been observed. It starts by checking that there is at least one target agent (line 2). If so, it identifies all actions that might have been executed by one target agent (lines 3-4). An action might have been executed if it is consistent according to the static properties, the domain knowledge, and the initial and final states. For each consistent action, it reconstructs the actions of the remaining agents recursively (lines 5-13).

Specialized Search (Algorithm 4). This algorithm is a factorial algorithm, which entails that the algorithm may not feasible even for reasonable-sized inputs.

Algorithm 4 Specialized Search

1: function SPECIALIZEDSEARCH(i, Act, TA)  
2: if $TA \neq \emptyset$ then  
3: for all $d \in D$ do  
4: if $\exists \sigma : \text{checkActionConsistency}(\sigma \cdot d, i, f)$ then  
5: $\alpha = \text{actor}(\sigma \cdot d)$  
6: $i' \leftarrow i \land \text{pre}(\sigma \cdot d)$  
7: $f' \leftarrow f \land \text{post}(\sigma \cdot d)$  
8: $Act' \leftarrow Act \land \sigma \cdot d$  
9: $TA' \leftarrow TA \setminus \{\alpha\}$  
10: $S' \leftarrow \text{search}(i', f', Act', TA')$  
11: for all $S_i \in S$ do  
12: $S_i \leftarrow S_i \cup \sigma \cdot d$  
13: return $S'$

Algorithm 5 Approximate Search.

Algorithm 5 Approximate Search.

1: function APPROXIMATESEARCH(i, Act, TA)  
2: if $TA \neq \emptyset$ then  
3: for all $d \in D$ do  
4: if $\exists \sigma : \text{checkActionConsistency}(\sigma \cdot d, i, f) \land \text{actor}(\sigma \cdot d) \in TA$ then  
5: $\alpha = \text{actor}(\sigma \cdot d)$  
6: $i' \leftarrow i \land \text{pre}(\sigma \cdot d)$  
7: $f' \leftarrow f \land \text{post}(\sigma \cdot d)$  
8: $Act' \leftarrow Act \land \sigma \cdot d$  
9: $TA' \leftarrow TA \setminus \{\alpha\}$  
10: $S' \leftarrow \text{search}(i', f', Act', TA')$  
11: for all $S_i \in S$ do  
12: $S_i \leftarrow S_i \cup \sigma \cdot d$  
13: return $S'$
Algorithm 4 Approximate Reconstruction Algorithm

```plaintext
1: function APPROXIMATERECONSTRUCTION(i, f, Act, TA, D)
2:   S ← approximateSearch(i, f, Act, TA)
3:   for all α ∈ TA do
4:     if |Sα| = 1 then
5:       R ← R ∪ Sα
6:     if R ≠ ∅ then
7:       Act ← Act ∪ R
8:       i ← i ∩ pre(R)
9:       if |Act| < |Ag| then
10:      f ← f ∩ post(R) ∧ i
11:     else
12:      f ← i ∩ post(Act)
13:     O, P ← calculateInstances(i)
14:   for all α ∈ TA do
15:     if |Sα| > 1 then
16:       if ∃a ∈ Sα : forbidden(P, a) then
17:         D ← D ∪ a
18:     else if ∃a ∈ Sα : mandatory(O, a) then
19:         D ← D ∪ a
```

Algorithm 5 Approximate Search Function

```plaintext
1: function APPROXIMATESEARCH(i, f, Act, TA)
2:   continue ← true
3:   for all α ∈ TA do
4:     Sα ← ∅  \hspace{1cm} \triangleright List of approximate actions per agent
5:   while continue ∧ TA ≠ ∅ do
6:     continue ← false.
7:   for all α ∈ TA do
8:     Lα ← ∅
9:   for all d ∈ D do
10:  if ∃σ : checkActionConsistency(σ · d, i, f) ∧
11:  actor(σ · d) ∈ TA then
12:    Lactor(σ · d) ← Lactor(σ · d) ∪ σ · d
13:  for all α ∈ TA do
14:   if |Lα| = 1 then
15:   Sα ← Lα
16:  i ← i ∩ pre(Lα)
17:  f ← f ∩ post(Lα)
18:  TA ← TA \ {α}
19:  continue ← true
20: for all α ∈ TA do
21:   Sα ← Lα
```

V. EVALUATION

This section covers the performance of a NM with full reconstruction, a NM with approximate reconstruction and a traditional norm monitor —which is the method used in the majority of previous proposals [8, 25, 16, 26, 11]— that only considers the observed actions to detect violations, with respect to their capabilities to monitor norm compliance. We have evaluated our proposal in a case study, which allows us to contextualise the results and to give a meaningful interpretation to them; and in a series of random experiments, which allow us to evaluate our proposal under a wide range of different situations and parameter values.

A. Case Study

We implemented in Java a simulator of the paper example in which robots attend requests in offices connected through corridors. Compliance with the collision avoidance norm is controlled by a monitor that observes surveillance cameras. In each simulation, we generate corridors and cameras randomly. In each step of the simulation, each robot chooses randomly one applicable action to be executed. The simulation is executed 100 steps and repeated 100 times to support the findings. We conducted experiments in which the number of offices \( O \) took a random value within the \([3, 500]\) interval and the number of robots \( R \) took a random value within \([2, 250]\) interval. Besides that, to be able to compare with the full NM, we also considered small scenarios only, in which the number of offices \( O \) takes a random value within \([3, 10]\) and the number of robots \( R \) takes a random value within \([2, 5]\), as the full reconstruction has an exponential cost and it is intractable for most of the cases with the default intervals.

1) Action Observability: To analyse the performance and scalability of monitors with respect to their capabilities to observe actions, we defined the number of corridors \( C \) as a random value within the \([O, O \times (O − 1)]\) interval and varied the ratio of cameras to corridors (action observability). Table I shows the percentage of violations detected per each type of monitor. The higher the ratio of cameras, the more actions are observed and the better the performance of all monitors. Moreover, the approximate NM offers on average a 39% performance improvement over a traditional monitor (i.e., it identifies 16% more violations plus a further 24% of discovered violations). That is, an approximate NM outperforms a traditional monitor with the same capabilities to observe actions. When compared to full NM in small scenarios \((O ∈ [3, 10]) and \(R ∈ [2, 5]\), approximate NM performs similarly. This is explained by the fact that there is a single norm in this scenario; actions have no concurrency conditions, and the preconditions and postconditions of actions are disjoint. In these circumstances, the approximate reconstruction process reconstructs actions similarly to the full reconstruction.

| Action Observability | Cameras Ratio | Traditional Monitor | Approximate NM | Approximate NM
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>20%</td>
<td>11%</td>
<td>14+9%</td>
<td>20%</td>
</tr>
<tr>
<td></td>
<td>40%</td>
<td>31%</td>
<td>40+12%</td>
<td>40%</td>
</tr>
<tr>
<td></td>
<td>60%</td>
<td>55%</td>
<td>71+10%</td>
<td>60%</td>
</tr>
<tr>
<td></td>
<td>80%</td>
<td>78%</td>
<td>91+4%</td>
<td>80%</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>100%</td>
<td>100+4%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table I: Action Observability Experiment

2) Action Instantiations: To analyse the performance and scalability of monitors with respect to agent capabilities to execute actions (i.e., the number of instantiations per action), we varied the ratio of corridors \([C] \) (e.g., a ratio of 0% means \( C = O \)) and defined the number of cameras as a random value within the \([0, C]\) interval. Table II shows the results of this experiment. The approximate NM offers on average a 43% performance improvement over a traditional monitor (i.e., it

\(^{18}\)Note that full reconstruction does not guarantee completeness.

\(^{19}\)Recall that \( C \) takes values within the \([0, O \times (O − 1)]\) interval.
identifies 29% more violations plus a further 14% of discovered violations). That is, given the same number of possible instantiations per action, an approximate NM outperforms a traditional monitor. Besides, we can see that, as in the previous experiment, the approximate NM performs similarly to the full NM. In particular, when the ratio of corridors is higher than 0%, agents are capable of executing different actions and the reconstruction process becomes more complex, which decreases the performance of full and approximate NMs. However, full and approximate NMs noticeably outperform the traditional monitor regardless of the ratio of corridors.

<table>
<thead>
<tr>
<th>Ratio</th>
<th>Traditional Monitor</th>
<th>Approximate NM Identify + Discover</th>
<th>Traditional Monitor</th>
<th>Full NM</th>
<th>Approximate NM Identify + Discover</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>51%</td>
<td>98+0%</td>
<td>0%</td>
<td>52%</td>
<td>99+0%</td>
</tr>
<tr>
<td>20%</td>
<td>48%</td>
<td>55+6%</td>
<td>20%</td>
<td>59%</td>
<td>79+3%</td>
</tr>
<tr>
<td>40%</td>
<td>48%</td>
<td>56+7%</td>
<td>40%</td>
<td>55%</td>
<td>74+3%</td>
</tr>
<tr>
<td>60%</td>
<td>41%</td>
<td>47+6%</td>
<td>60%</td>
<td>51%</td>
<td>70+4%</td>
</tr>
<tr>
<td>80%</td>
<td>49%</td>
<td>56+13%</td>
<td>80%</td>
<td>55%</td>
<td>68+5%</td>
</tr>
<tr>
<td>100%</td>
<td>42%</td>
<td>49+7%</td>
<td>100%</td>
<td>57%</td>
<td>70+4%</td>
</tr>
</tbody>
</table>

TABLE II: Action Instantiations Experiment

B. Random Experiments

We implemented a simulator in Java in which there is a set of agents that perform actions in a monitored environment as defined below. In particular, our simulator does not model a specific scenario; rather it creates a different scenario in each simulation (i.e., generating randomly agent capabilities, the environment properties, actions and norms). As in the previous experiments, we have considered big and small scenarios. In particular, the number of agents $G$ in small scenarios took a random value within the $[1, 5]$ interval, whereas in big scenarios $G$ took a random value within the $[1, 500]$ interval. The number of actions $A$ took a random value within the $[1, 50]$ interval. Again, the simulation is executed 100 steps and repeated 1000 times to ensure that the values of the simulation parameters range over possible values.

Agent Definition. We modelled different types of agents with different capabilities to perform actions. In particular, the set of actions available to each agent depends on the function(s) assumed by each agent in a particular simulation. To model these capabilities, a set of roles is created at the beginning of each simulation. Specifically, the number of roles created took a random value within the $[1, A]$ interval. For each role a subset of the actions are randomly selected as the role capabilities; i.e., all agents enacting this role are able to perform these actions. To avoid that all roles have similar capabilities, which would lead to simulations populated by homogeneous agents, the number of actions selected as role capabilities took a random value within the $[1, 0.1 \times A]$ interval (i.e., at maximum each role is capable of performing a 10% of the actions). At the beginning of each simulation, each agent is defined as enacting a random subset of the roles. In each step of the simulation, each agent selects randomly one action among the available actions that it can execute in the current state.

Environment Definition. In the simulator, the environment is described in terms of different situations or states of affairs that can be true or false. Each one of these states of affairs is represented by a grounded proposition. Thus, the state of the environment is defined in terms of a set of propositions. For simplicity, we assumed that these propositions are independent (i.e., propositions are not logically related). In our simulations, the number of propositions $P$ took a random value within the $[A, 2 \times A]$ interval (i.e., there is at least one proposition per each action). Besides that, there is a set of grounded atomic formulas describing the roles played by agents and the actions that can be performed by each role. The relationship between agents and roles is formally represented by a binary predicate $(play)$. Specifically, the expression $play(g, r)$ describes the fact that the agent identified by $g$ enact the role identified by $r$. Similarly, relationship between roles and actions is formally represented by a binary predicate $(capable)$. Specifically, the expression $capable(a, r)$ describes the fact that agents enacting role $r$ are capable of performing the action identified by $a$. For simplicity, we assume that the roles enacted by the agents and the role capabilities are static properties of the environment.

Action Definition. Actions allow agents to change the state of the environment. At the beginning of each simulation, a set of actions is randomly generated. For each action $(name, pre, con, post)$ the elements are defined as follows: $name$ is initialised with a sequential identifier $a$; $pre$ is defined as $\{play(A, R), capable(R, a), p_1, ..., p_n\}$ where the elements $p_1, ..., p_n$ are randomly selected from the proposition set; $con$ is defined as $\{a_1(A_1, R_1), ..., a_m(A_m, R_m)\}$, where each $a_i$ is an action randomly selected from the action set such that $a_i \neq a$ and $A_i, R_i$ are free variables representing the agent performing the action and the role capable of performing this action, respectively; and $post$ is defined as $\{p_1, ..., p_k\}$ where each $p_i$ is a proposition randomly selected from the proposition set. To avoid that actions have too many constraints, which would be unrealistic and make actions to be only executed on few situations, the number of propositions in $pre$ and $post$ takes a random value within the $[1, [0.1 \times P]]$ interval. Similarly, the number of actions in $con$ takes a random value within the $[0, [0.1 \times A]]$ interval.

Besides these actions, a NOP action, which has no effect on the environment, was created. Our simulator models scenarios where the NOP action can always be observed. This is the case in many real domains such as Intrusion Detection Systems or Autonomous Systems, where it is not always possible to analyse the data (e.g., the packages) sent by agents (e.g., hosts) to infer the actions performed, but it is always possible to know which agents have performed an action (i.e., which agents have sent packages). To maximise the number of actions executed in the simulations, which may entail more violations and fulfilments, we defined that the NOP action can only be executed by agents when none of their available actions can be

---

20Note that in the random experiments there are more simulation parameters than in the case-study simulator and a higher number of repetitions is required to support the findings.  
21This condition has been formulated in action preconditions as explained below.  
22Note that an action can change the truth value of several propositions.
executed. However, similar results would have been obtained if this condition was relaxed.

**Norm Definition.** Agents’ actions are regulated by a set of norms. At the beginning of each simulation, a set of norms is randomly created. In particular the number of norms took a random value within the \([1, A]\) i.e., there is at maximum one norm per each action). For each norm \((\text{deontic, condition, action})\) the elements are defined as follows: \text{deontic} is randomly initialised with a deontic operator; \text{condition} is defined as \(\{p_1, ..., p_k\}\) where each \(p_i\) is a proposition randomly selected from the proposition set; and \text{action} is randomly initialised with an action. To allow norms to be instantiated, the number of propositions in \text{condition} takes a random value within the \([0, [P * 0.1]]\) interval.

1) **Action Observability:** To analyse the performance and scalability of monitors with respect to their capabilities to observe actions, we varied the observation probability. Tables III and IV show the percentage of detected fulfilments and violations, respectively. Again, the approximate NM offers a significant performance improvement over a traditional monitor; i.e., the approximate NM offers on average a 74% performance improvement over a traditional monitor. When compared to full NM in small scenarios \((A \in [1, 50])\) and \(G \in [1, 5]\), the full NM offers on average a 21% performance improvement over an approximate NM. This is explained by the fact that this experiment is more complex than the case study; i.e., there are several norms (both prohibition and obligation norms), actions have concurrent conditions and actions may have conflicting preconditions and postconditions (i.e., conditions that are defined over the same propositions). Note that the traditional monitor detects violations and fulfilments even when the observation probability is 0%. These detections correspond to situations in which none of the agents can execute any action (i.e., all agents execute the NOP action) which leads to the fulfilment of prohibition instances and the violation of obligation instances. This phenomenon is more frequent in case of small scenarios since the lower the number of agents, the higher the probability that all agents cannot execute any action.

<table>
<thead>
<tr>
<th>Actions</th>
<th>Traditional Monitor</th>
<th>Approximate NM</th>
<th></th>
<th>Actions</th>
<th>Traditional Monitor</th>
<th>Approximate NM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Identify+Discover</td>
<td></td>
<td></td>
<td></td>
<td>Identify+Discover</td>
</tr>
<tr>
<td>2</td>
<td>52%</td>
<td>95+4%</td>
<td></td>
<td>2</td>
<td>69%</td>
<td>99%</td>
</tr>
<tr>
<td>8</td>
<td>49%</td>
<td>75+1%</td>
<td></td>
<td>8</td>
<td>47%</td>
<td>73%</td>
</tr>
<tr>
<td>32</td>
<td>32%</td>
<td>48+0%</td>
<td></td>
<td>32</td>
<td>36%</td>
<td>63%</td>
</tr>
<tr>
<td>128</td>
<td>29%</td>
<td>40+0%</td>
<td></td>
<td>128</td>
<td>53%</td>
<td>83%</td>
</tr>
</tbody>
</table>

**TABLE IV: Violations Detected in the Action Observability Experiment**

2) **Action Possibilities:** To analyse the performance and scalability of monitors with respect to agent capabilities to execute actions (i.e., the number of actions available), we defined the observation probability as a random value within the \([0\%, 100\%]\) interval and we varied the number of actions.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>2%</td>
<td>35+0%</td>
<td></td>
<td>0%</td>
<td>20%</td>
<td>46%</td>
<td>34+1%</td>
</tr>
<tr>
<td>20%</td>
<td>18%</td>
<td>47+4%</td>
<td></td>
<td>20%</td>
<td>23%</td>
<td>51%</td>
<td>39+1%</td>
</tr>
<tr>
<td>40%</td>
<td>35%</td>
<td>72+3%</td>
<td></td>
<td>40%</td>
<td>31%</td>
<td>57%</td>
<td>45+1%</td>
</tr>
<tr>
<td>60%</td>
<td>50%</td>
<td>80+2%</td>
<td></td>
<td>60%</td>
<td>43%</td>
<td>69%</td>
<td>56+0%</td>
</tr>
<tr>
<td>80%</td>
<td>60%</td>
<td>80+2%</td>
<td></td>
<td>80%</td>
<td>58%</td>
<td>79%</td>
<td>70+0%</td>
</tr>
<tr>
<td>100%</td>
<td>100%</td>
<td>100+0%</td>
<td></td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100+0%</td>
</tr>
</tbody>
</table>

**TABLE III: Fulfilments Detected in the Action Observability Experiment**

\[^{23}\text{Note that when the NOP action is always observable, the capabilities of monitors to detect violations and fulfilments do not depend on the fact that agents are allowed to perform the NOP action in any situation.}\]

3) **Actions:** To analyse the performance and scalability of monitors with respect to agent capabilities to execute actions (i.e., the number of actions available), we defined the observation probability as a random value within the \([0\%, 100\%]\) interval and we varied the number of actions.

| Actions | Traditional Monitor | Approximate NM | | Actions | Traditional Monitor | Full NM | Approximate NM |
|---------|---------------------|----------------|---|---------------------|----------|----------------|
| 2       | 33%                 | 95+5%          | | 2       | 68%                 | 99%        |
| 8       | 49%                 | 76+6%          | | 8       | 48%                 | 76%        |
| 32      | 35%                 | 50+1%          | | 32      | 38%                 | 62%        |
| 128     | 31%                 | 39+0%          | | 128     | 56%                 | 83%        |

**TABLE V: Fulfilments Detected in the Action Possibilities Experiment**

\[^{23}\text{Note that when the NOP action is always observable, the capabilities of monitors to detect violations and fulfilments do not depend on the fact that agents are allowed to perform the NOP action in any situation.}\]

C. Summary

The conclusions of our evaluation are threefold:

1) Both approximate and full reconstruction processes are more effective (i.e., detect more norm violations and fulfilments) than traditional monitoring approaches regardless of the scenario complexity (i.e., action possibilities and observability). Both in the case study and in the random experiments our algorithms improved significantly the percentage of violations and fulfilments detected.

2) Approximate reconstruction is slightly less effective than full reconstruction. In the case study, where a single
Influence on agent behaviours. These are classified into two main categories: regimentation mechanisms, which make the violation of norms impossible; and enforcement mechanisms, which are applied after the detection of norm violations and fulfilments, reacting upon them.

Regimentation mechanisms prevent agents from performing forbidden actions (vs. force agents to perform obligatory actions) by mediating access to resources and the communication channel, such as Electronic Institutions (EIs). However, the regimentation of all actions is often difficult or impossible. Furthermore, it is sometimes preferable to allow agents to make flexible decisions about norm compliance. In response to this need, enforcement mechanisms were developed. Proposals on the enforcement of norms can be classified according to the entity that monitors whether norms are fulfilled or not. Specifically, norm compliance can be monitored by either agents themselves or the underlying infrastructure may provide monitoring entities.

Regarding agent monitoring, this approach is characterized by the fact that norm violations and fulfilments are monitored by agents that are involved in an interaction, or other agents that observe an interaction in which they are not directly involved. The main drawback of proposals based on agent monitoring is the fact that norm monitoring and enforcement must be implemented by agent programmers.

Regarding infrastructural monitoring, several authors proposed developing entities at the infrastructure level that are in charge of both monitoring and enforcing norms. Cardoso & Oliveira proposed an architecture in which the monitoring and enforcement of norms is made by a single institutional entity. This centralized implementation represents a performance limitation when dealing with a considerable number of agents. To this limitation of centralized approaches, distributed mechanisms for an institutional enforcement of norms were proposed.

All of the aforementioned proposals on monitoring assume that monitors have complete observational capabilities. Exception to these approaches is the recent work of Bulling et al. and Alechina et al. In bulling the partial observability problem is addressed combining different norm monitors to build ideal monitors (i.e., monitors that together are able to detect the violation of a given set of norms). In alechina the authors propose to synthesise an approximate set of norms that can be monitored given the observational capabilities of a monitor. However, there are circumstances in which norms cannot be modified (e.g., contract and law monitoring) or ideal monitors are expensive and/or not feasible. We take a different approach in which norms and monitors’ observation capabilities remain unchanged and monitors reconstruct unobserved actions.

Our approach is also related to planning, where methods (e.g., POMDPs) for choosing optimal actions in partially observable environments have been proposed. A major difference between these proposals and our proposal is that NMs do not perform practical reasoning, i.e., they do not try to optimise or achieve a practical goal. Instead, NMs perform both deductive and abductive reasoning to reason from observed actions to reach a conclusion about the state of the world, and to infer unobserved actions.
In this paper, we propose information models and algorithms for monitoring norms under partial action observability by reconstructing unobserved actions from observed actions using two different reconstruction processes: full and approximate. Our experiments demonstrate that both reconstruction processes significantly detect more norm violations than traditional monitoring approaches. Approximate reconstruction performs slightly worse than full reconstruction, whereas its computational cost is much cheaper, making it suitable to be applied in practice.

The reconstruction algorithms proposed in this paper can be applied to several domains that require action monitoring; from normative MAS, to intrusion detection systems, to control systems and to intelligent surveillance systems. As future work, we plan to investigate domain-dependent approximations that could speed up action reconstruction.

REFERENCES


Natalia Criado is Senior Lecturer in the School of Computing and Maths at Liverpool John Moores University (UK), which she joined in 2014. Her main research interests are on the intersection between Artificial Intelligence and Normative Systems, and in particular, intelligent/automated approaches to norm reasoning, with a strong focus on Multi-agent Systems and BDI Agents.

Jose M. Such is Lecturer (Assistant Professor) in the School of Computing and Communications at Lancaster University (UK), which he joined in 2012. His main research interests are on the intersection between Artificial Intelligence, Cyber Security and Human Factors.