Autonomous Agents and Multi-Agent Systems (2023) 37:46
https://doi.org/10.1007/s10458-023-09627-4

A normative approach for resilient multiagent systems

Geeta Mahala[1] · Özgür Kafalı[2] · Hoa Dam[1] · Aditya Ghose[1] · Munindar P. Singh[3]

Accepted: 9 October 2023 / Published online: 13 November 2023

Abstract
We model a multiagent system (MAS) in socio-technical terms, combining a social layer consisting of norms with a technical layer consisting of actions that the agents execute. This approach emphasizes autonomy, and makes assumptions about both the social and technical layers explicit. Autonomy means that agents may violate norms. In our approach, agents are computational entities, with each representing a different stakeholder. We express stakeholder requirements of the form that a MAS is resilient in that it can recover (sufficiently) from a failure within a (sufficiently short) duration. We present ReNo, a framework that computes probabilistic and temporal guarantees on whether the underlying requirements are met or, if failed, recovered. ReNo supports the refinement of the specification of a socio-technical system through methodological guidelines to meet the stated requirements. An important contribution of ReNo is that it shows how the social and technical layers can be modeled jointly to enable the construction of resilient systems of autonomous agents. We demonstrate ReNo using a manufacturing scenario with competing public, industrial, and environmental requirements.

Keywords Norms · Multiagent systems · Resilience

✉ Geeta Mahala
gm168@uowmail.edu.au

Özgür Kafalı
ozgurkafali@gmail.com

Hoa Dam
hoa@uow.edu.au

Aditya Ghose
aditya@uow.edu.au

Munindar P. Singh
mpsingh@ncsu.edu

1 School of Computing and Information Technology, University of Wollongong, Wollongong, NSW, Australia

2 School of Computing, University of Kent, Canterbury, Kent, UK

3 Department of Computer Science, North Carolina State University, Raleigh, NC, USA

1 Introduction

Models of social interaction are central to artificial intelligence (AI) and have long drawn interest from the research community, especially in the field of multiagent systems (MAS) [9, 20, 40, 52]. Socio-technical systems (STS) are a powerful way of expressing social interaction among agents and provide a way of governing MAS [11, 29]. In a socio-technical system (STS), autonomous agents act on behalf of the stakeholders and represent their needs [13, 17, 26]. We adopt a conception of an STS [27, 28] in which agents interact with each other and with the underlying technical architecture. Such an STS can be represented as a multiagent system (MAS) and governed via norms [51] that regulate interactions among the agents and help guide the agents towards the achievement of their stakeholders' needs.

This paper falls into the broad area of socio-technical systems. It goes beyond previous research by introducing the notion of resilient socio-technical systems. A resilient STS is one that seeks to meet stakeholder requirements and can recover from requirement failures. In this way, resilience is an essential element of the trustworthiness of an STS, especially in regards to its ability [54].

Our intuitive reading of the resilience of an STS takes the following form: *Within some deadline, if an undesirable condition, or a condition that flags a departure from normal operating states, comes to pass, then within k steps a desirable state (or one that represents a return to normal operating conditions) can be achieved with a probability higher than a specified threshold.* We identify three important aspects of resilience: (1) Resilience involves recovery from an undesirable state; (2) Resilience involves recovery within a deadline (or a pre-specified number of steps); (3) Resilience involves recovery with a probability exceeding some (sufficiently high) threshold.

We tackle the above intuitions by extending STS specifications to include time and quantities, providing a formalization and algorithm for automatically translating STS specifications to models that can be input into model-checking tools such as PRISM [30], introducing probabilistic model-checking to STS specifications to reason about the probability of meeting requirements, providing a means to evaluate trade-offs between achieving technical objectives and meeting social regulations, and finally, implementing a prototype of the entire framework.

1.1 Research objectives and contributions

Our *goal* is to aid the design of resilient STSs by providing a technique and tooling for evaluating alternative STS specifications with respect to the stated requirements. Specifically, we have the following objectives:

- O_1 To incorporate a rich model (time and quantities) of computational norms into STS specifications as a means to regulate agent behavior and guide which actions an agent should take.
- O_2 To verify at a specified level of likelihood that an STS meets its stakeholders' requirement and can recover from a requirement failure within a specified period of time.
- O_3 To develop techniques (supported by methodological guidelines) for leveraging probabilistic model checking to explore the trade-off space involving alternative STS designs and alternative formulations (particularly relaxations) of stakeholder requirements.

Accordingly, we present ReNo (short for Resilience via Norms), a probabilistic framework that evaluates how resilient an STS specification is by verifying to what extent the MAS meets its stakeholders' requirements and evaluating the speed and extent to which the MAS recovers from a requirement failure.

Our contributions include (i) a formal STS specification including social norms and technical actions, and associated requirements expressed with quantities and time—achieves O_1; (ii) a transformation algorithm that takes a given STS specification and associated requirements, and produces a PRISM (Probabilistic Symbolic Model Checker) [30] model and associated properties in Probabilistic Computation Tree Logic (PCTL) [1, 25]—provides the means to achieve O_2; and (iii) a formal probabilistic verification process stating how likely a given STS specification meets stakeholder requirements and recovers from requirements failures—achieves O_2 and O_3.

1.2 Practical usage, as envisioned

We envision ReNo being deployed in practice via the following steps. First, the designer specifies an initial version of the STS specification. Second, the stakeholder provides the requirements expressed in PCTL. The requirements may include resilience requirements (formalized in Sect. 3 as system properties) where we show how the specified MAS world recovers from requirement failures. Third, ReNo generates a PRISM model from the given STS specification. Fourth, the designer runs the verification process. Fifth, ReNo produces an output demonstrating how likely the STS specification meets the stated requirements. Sixth, the designer refines the STS specification based on the output produced by ReNo. The process iterates until the requirements are satisfied. If there is a situation where the requirements are not satisfied, the stakeholders would have the option to relax the requirements suitably. Seventh, the STS designer and stakeholders can analyse the various state variables or other parameters to produce a new relaxed requirement based on the output produced by ReNo.

1.3 Organization

The rest of the document is organized as follows. Section 2 defines socio-technical systems (STSs) and introduces the relevant background. Section 3 describes the computational elements of ReNo. Section 4 describes the methodological guidelines. Section 5 presents our prototype implementation and verification experiments. The complete replication package, which encompasses all the artifacts and experimental data, including the outcomes of actions and their corresponding execution probabilities can be accessed at [34]. Section 6 describes the related work. Section 7 concludes the paper.

2 Background

2.1 Socio-technical systems

Our overarching objective is to develop a framework for handling resilience requirements in multiagent systems (MAS) composed of agents that are afforded autonomy and self-interest and where these interests might conflict with those of other agents. This notion is a departure from traditional work on engineering multiagent systems, where a significant body of work assumes that the agents involved have the same interests and have limited autonomy [55].

An important challenge in such settings is managing the aggregate behaviour of the participating agents and, in particular, ensuring that the aggregate behaviour meets certain requirements imposed on the MAS. One effective strategy for addressing this challenge involves incorporating norms. Norms constrain the behavior of individual agents in group settings (e.g., societies and communities), and regulate the interactions between those individuals. Norms in multiagent systems specify social controls on an agent's actions and can help achieve the overall objectives of the system.

We conceive of a multiagent system in normative terms and, in particular, as a socio-technical system (STS) consisting of a social and a technical layer.

The technical layer of an STS is composed of software components supporting various agent actions. The social layer comprises the stakeholders and, for our purposes, the agents who represent those stakeholders. Norms regulate the interactions among the entities (agents and humans) in the social layer. For our purposes, the stakeholders are not formally modeled and the agents (which are formally modeled) capture all the relevant actions. Specifically, the norms here are directed from one agent to another and state what one agent may legitimately expect of another and under what conditions. That is, the agents, being autonomous parties, can violate any norm that applies to them but if they do so, they are identified as being in violation of the norm. Sometimes, an agent must violate a norm to achieve a greater objective [50]. The technical layer (actions) and social layer (norms) are specified by an STS designer, in accordance with stakeholders' requirements.

The distinction between the social and technical layers is important throughout the technical development we present below. The Methodological guidelines (in Sect. 4) are provided that support the design and implementation of resilient STS. In methodological guidelines, it is demonstrated that updating the STS can involve changes to both the social and technical layers in order to meet stakeholder requirements. Changes at the technical layer involve substantial changes to agent capabilities while changes in the social layer involve changes to the norms guiding agent interactions. For example, if a given STS specification does not meet the stated requirements then either the STS specification can be refined by adding or removing agents' actions at the technical layer or adding or removing norms at the social layer.

To explain what we mean by an STS, let's describe a simple use case that illustrates how the social and technical layers arise and interplay in a practical multiagent scenario.

Example of an STS with agents, norms, and software components

Consider a scenario involving the manufacture of personal protective equipment (PPE) where different stakeholders have potentially conflicting functional and sustainability requirements. Imagine there are two textile companies for manufacturing PPE units, one (*CompanyNear*) located near a population center and the other company (*CompanyFar*) located far from the population center. This STS has four agents: (1) a textile manufacturing firm *CompanyNear*; (2) another textile manufacturing firm *CompanyFar*; (3) a *Hospital* with a requirement for varying quantities of PPE units (this demand can go up if there is an outbreak of a virus such as Covid-19); (4) an environmental *Regulator* that imposes rules governing the extent of permitted pollution and which it enforces through occasional inspections.

The companies have different kinds of environmental impacts. The *regulator* imposes stricter prohibitions on CompanyNear relative to CompanyFar since CompanyNear is located near the city and pollution generated by it will have a potentially greater adverse impact on the health of city residents. These norms govern this STS:

- A norm governing the amount of pollution *CompanyNear* is permitted to generate.

- A similar norm governing the amount of pollution *CompanyFar* is permitted to generate (which, in general, can be higher than for *CompanyNear* since *CompanyFar* is farther from the population center).
- Norms governing the quantities of PPE that *CompanyNear* and *CompanyFar* are committed to producing for the *Hospital* (in our example, both companies commit to producing equal quantities of PPE, but these amounts could be different in general).

Both *CompanyNear* and *CompanyFar* can *act* to manufacture varying quantities of PPE units (we provide details on how these actions are represented in Sect. 3.3). The *Hospital* agent performs only one action which involves creating demand for PPE units, which is manifested via the commitments that *CompanyNear* and *CompanyFar* make to the *Hospital*. In order to mitigate the potential increase in pollution associated with increased production of PPE units. The *CompanyNear* and *CompanyFar* agents execute actions that involve decreasing pollution. These actions are manifested via the commitments that *CompanyNear* and *CompanyFar* make to the *Regulator*. The *Regulator* agent performs a variety of actions including specifying regulatory limits on the permitted pollution level for each company, occasional monitoring of pollution levels, and fining firms that violate pollution limits. For the purposes of our example, we focus on the specification of regulatory limits on pollution, and this action manifests only in the form of prohibitions that the PPE manufacturers must abide by.

In contrast to norms, requirements are distinct as they are specific conditions or constraints that must be met to achieve a particular goal or objective. In our work, we evaluate whether the stated requirement will be satisfied based on the norms and actions involved. Given the above STS specification, we can assert a variety of requirements. An example requirement is that both *CompanyNear* and *CompanyFar* should produce 1000 units of PPE per week and both companies should not exceed a pollution level of 50 ppm (parts per million). Another example requirement, regarding the resilience of the STS, is that both *CompanyNear* and *CompanyFar* should reduce the pollution level from above 100 ppm (an unacceptable level of pollution) to below 60 ppm (an acceptable level of pollution) of the chemical in question.

2.2 PCTL

Our goal is to create a practical framework to govern multiagent systems that meet specified resilience requirements (along with functional requirements). A practical approach to specifying such requirements is to state them in probabilistic rather than absolute terms. With the probability as a tunable parameter set to a high value, it is possible to require that a system be resilient most of the time even if not all the time.

There is a growing realization in the literature that for reasons similar to those that make it worthwhile to consider actions with uncertain outcomes, it is important to support probabilistic goals [33]. Properties such as resilience are especially amenable to a probabilistic specification since they do not concern the object-level behaviour of the system (even if these must meet hard, non-probabilistic, requirements).

Our framework emphasises STS specifications to understand the probability of meeting requirements and recovering from a requirement failure within a specified time. PCTL is used for specifying properties of discrete-time models such as discrete-time Markov chains (DTMCs) [31]. In ReNo, a PRISM model is generated from the given STS specification and is also a discrete-time model, whose state space is discrete and transitions are discrete steps between states. Therefore, we use temporal PCTL to specify the properties of the system to be checked.

We operationalize an STS via Probabilistic Computation Tree Logic (PCTL). PCTL is derived from CTL and includes a probabilistic operator $\mathcal{P}$ [25]. PCTL is equipped with temporal operators with time bounds where time is discrete and one *time unit* corresponds to one transition along an execution path. In PCTL, each atomic proposition is a *state formula*, which involves assertions that are true in a single state (i.e., *state properties*). *Path formulas* describe properties of paths (i.e., sequences of states). Formally, state and path formulas in PCTL are defined as follows:

$$\begin{aligned} &\textit{State formulas} \quad \phi ::= true \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \mathcal{P}_{\sim prob}(\Psi) \\ &\textit{Path formulas} \quad \Psi ::= X\phi \mid \phi \mathsf{U}^{\leq t} \phi \mid \phi \mathsf{U} \phi \end{aligned}$$

Where a is an atomic proposition and Ψ represents a path formula, ϕ represents a state formula and $\mathcal{P}$ is a probabilistic operator (details below), $\sim$ is an inequality (i.e., $\sim \in \{<, \leq, >, \geq\}$), and $prob \in [0, 1]$ is a probability bound and $t \in \mathbb{N}$.

Path formulas Ψ use the Next (X), Bounded Until $\mathsf{U}^{\leq t}$, and Unbounded Until (U) operators. Similar to "always $\Box$" and "eventually $\Diamond$" operators in CTL, there are temporal operators in PCTL [1]. In the literature, the temporal operators $\Box$ and G have been interchangeably used for "always" and $\Diamond$ and F have been interchangeably used for "eventually". Here, we use G and F to align with PRISM syntax.

$$\begin{aligned} [F^{\leq t}\phi]_{\sim p} &\equiv [true \mathsf{U}^{\leq t}\phi]_{\sim p} \\ [G^{\leq t}\phi]_{\sim p} &\equiv \neg[true \mathsf{U}^{\leq t}\neg\phi]_{\sim(1-p)} \end{aligned}$$

A state formula is used to express the property of a model. A path formula may occur only as the parameter of the probabilistic path operator $\mathcal{P}_{\sim prob}(\Psi)$. Intuitively, a state s satisfies $\mathcal{P}_{\sim prob}(\Psi)$ if the probability of taking a path from s satisfying Ψ is in the interval specified by $\sim prob$.

PRISM is a probabilistic model checker [30]. Properties of the system are represented formally in a probabilistic temporal logic (such as PCTL) and automatically verified against an input probabilistic state transition model. PRISM takes two inputs: (i) a *probabilistic model* (such as a Markov decision process (MDP)) and (ii) a *property specification*. PRISM then conducts model checking to determine which states of the system satisfy the specification. PRISM supports path properties that can be used inside the $\mathcal{P}$ operator such as $[\mathsf{F}^{\leq 10} q]_{\geq 0.6}$ states that with at least 60% probability q will become true within 10 time units, while $[\mathsf{G}^{\leq 20}\, r]_{\geq 0.99}$ states that with at least 99% probability r will hold at least for 20 time units.

The syntax of the PRISM property specification language includes various probabilistic temporal logics, including PCTL, within PRISM. In the informal PRISM property specification language, a property is written as $\mathcal{P}_{bound}[pathprop]$, stating that the probability of being satisfied by the paths from the current state meets the bound $bound$.

An example of a bound would be $\mathcal{P}_{0.75}[pathprop]$ which means that the probability that the path property $pathprop$ is satisfied by the paths from state s is greater than 0.75. To consider the nondeterministic behaviour of a system then the meaning of $\mathcal{P}_{bound}[pathprop]$ is that the probability of $pathprop$ being satisfied by the paths from the current state meets the bound $bound$ for all possible resolutions of the nondeterminism. Therefore, we need to reason about minimum and maximum probability over all the possible resolutions of the nondeterminism. The minimum and maximum probabilities are calculated using $\mathcal{P}_{min=?}[pathprop]$ and $\mathcal{P}_{max=?}[pathprop]$, respectively. Additionally, we need to include parentheses when using logical operators in a path property ($pathprop$) because logical operators have precedence over temporal ones. An example of a property is $\mathcal{P}_{0.90}[(G\ \text{“}A''){\&}(F\ \text{“}B''))]$.

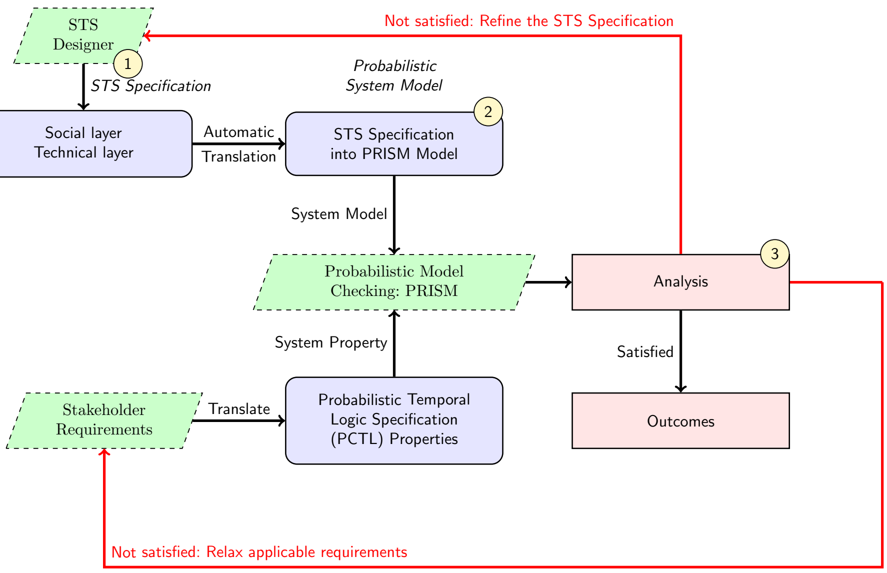
STS Designer
1
STS Specification
Social layer
Technical layer
Automatic Translation
Probabilistic System Model
2
STS Specification into PRISM Model
System Model
Probabilistic Model Checking: PRISM
Analysis
3
Not satisfied: Refine the STS Specification
Satisfied
Outcomes
System Property
Stakeholder Requirements
Translate
Probabilistic Temporal Logic Specification (PCTL) Properties
Not satisfied: Relax applicable requirements

Fig. 1 The ReNo Framework

3 The ReNo framework

Figure 1 shows ReNo's main components, each of which is explained below:

Stakeholders Stakeholders define the requirements that the STS must satisfy. How they do so is not in our present scope but can involve a combination of creativity [37] or argumentation [39]. If it turns out that satisfying a given set of requirements is not possible, then the stakeholders would have little choice but to relax their requirements (ideally as little as possible) to make it feasible to design an STS that satisfies these (relaxed) requirements. The reason for this, e.g., the number of steps being too small, is something that is hard to know ahead of time and is explored under the methodological guidelines as an iterative process in Sect. 4. In addition to the usual achievement and maintenance requirements, we introduce a new class of *resilience requirements*. Using the syntax of PCTL described above, a resilience requirement takes the following general form:

$$\langle UStateCond\rangle \rightarrow P_{\langle ineq\rangle probabilisticConst}[F^{\langle ineq\rangle integerConst}\langle DStateCond\rangle]$$

Here $\langle ineq\rangle$ is an operator (such as $\leq, <, \geq, or\ =$). $probabilisticConst$ is any real number between 0 and 1, representing a probability value. Here, $integerConst$ is, as the name suggests, an integer constant, used to denote the number of steps. $UStateCond$ abbreviates a condition representing an undesirable state (such as a state in which the level of pollution exceeds an acceptable threshold). $DStateCond$ abbreviates a condition representing a desirable state (such as one in which pollution levels are below an acceptable threshold). The generic resilience requirement thus states that if an undesirable state (denoted by $UStateCond$) transpires, then the system can return to a desirable state (denoted by $DStateCond$) at some future state but before or after, depending on the nature of the inequality, the system has transitioned through $integerConst$ states, with a probability that satisfies $\langle ineq\rangle probabilisticConst$. The above equation captures the essence of resilience requirements, ensuring that the system has both the timing and the probability factors considered to facilitate the transition from an undesirable state to a desirable state. It provides a quantifiable measure for assessing the system's ability to recover and adapt in the face of undesired conditions. We provide concrete examples of resilience requirements later in the paper.

STS Designer The designer specifies an STS as two layers. The *social layer* describes the agents and the norms among them, whereas the *technical layer* provides the operational actions by which the agents act. The social layer consists of a set of norms that govern the interactions among the agents. Note that norms [53] in our model are directed from one party to another. In our example, each PPE manufacturer would commit to the *Regulator* to meet certain pollution standards. That is, the norms are pair-wise, even with multiple agents. Actions in the technical tier allow or restrict specific agent actions as they represent hard constraints. Actions describe relevant facts about the operating environment, e.g., what will (potentially) happen when an action is executed.

Translation of an STS Specifcation to a PRISM Model RENO takes the STS specification as input to generate a PRISM model. A PRISM model is a probabilistic state transition model. A probabilistic state transition model associates a *transition probability* with each transition. What we generate is a small variation in the form of *augmented probabilistic state transition models* explained in Sect. 3.3.1. In this case, an augmented probabilistic state transition model takes into account both the probability of selecting an action and the probability of executing an action. The process of translating an STS specification into a PRISM model is explained in Sect. 3.3.2.

Analysis RENO verifies the STS against the requirements to understand how the STS would fare, i.e., with what probability it would violate or satisfy each requirement. If the analysis suggests that refinement is required, the STS designer leverages this understanding to refine the STS specification. If the analysis determines that it is not possible to modify the STS in a way that would satisfy the requirements, the stakeholder proceeds to analyze the various state variables or parameters to identify relaxed requirements, as explained in Sect. 4.2.

3.1 RENO Syntax

The RENO uses a language for specifying norms and actions. Table 1 shows RENO's syntax for specifying STSs. $\mathbb{AG} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ is a finite set of agents and $\Phi = \{\text{Attr}_1, \text{Attr}_2, \ldots, \text{Attr}_m\}$ is a finite set of attributes. Here, attributes are variables that are assigned values or appear in inequalities used in action specifications or in the antecedent and consequent conditions used in the various norm types (*PPE* and *pollution* in the discussion below are examples of attributes). A superscript of $+$ indicates one or more repetitions whereas superscript $*$ indicates zero or more. The operator $+=$ and $-=$ can be best understood with the following example: x$+=$ y updates x by adding y to its current value while x$-=$y updates x by subtracting y from its current value. If y is a numeric range (NRange) as opposed to a single value, then we update x by adding (resp. subtracting) a value chosen from NRange. L_i represents the line number corresponding to the ith line in Table 1. Table 1 begins with the core concepts of RENO and ends with primitive concepts such as Num.

L_8 Stmt can be either the update of the value of Attr using them $+=$ or $-=$ operators, the assignment of a value val to Attr or a reference to Attr in isolation. For example, if Stmt is PPE+=[100,220] where PPE is an attribute, then the current value of PPE is increased by adding a value between 100 and 220.

Table 1 ReNo Syntax

L_1	Specification	$\longrightarrow$	$\text{Norm}^+ \mid \text{Action}^+$
L_2	Norm	$\longrightarrow$	Commitment | Prohibition
L_3	Commitment	$\longrightarrow$	$c(\mathbb{AG}, \mathbb{AG}, \text{Cond}, \text{Cond})$
L_4	Prohibition	$\longrightarrow$	$p(\mathbb{AG}, \mathbb{AG}, \text{Cond}, \text{Cond})$
L_5	Action	$\longrightarrow$	$m(\text{Cond}, \text{Stmt}^+, \text{Stmt}^+)$
L_6	Cond	$\longrightarrow$	$\text{Expr} \mid \text{Cond} \wedge \text{Cond} \mid \text{Cond} \vee \text{Cond} \mid \sim \text{Cond}$
L_7	Expr	$\longrightarrow$	$\top \mid \bot \mid \text{Attr} \geq \text{Num} \mid \text{Attr} \leq \text{Num} \mid \text{Attr} = \text{Val}$
L_8	Stmt	$\longrightarrow$	$\text{Attr} \mathrel{+}= \text{NRange} \mid \text{Attr} \mathrel{-}= \text{NRange} \mid \text{Attr} = \text{Val} \mid \text{Attr}$
L_9	NRange	$\longrightarrow$	Num | [Num, Num]
L_{10}	Val	$\longrightarrow$	$\text{Num} \mid \top \mid \bot$
L_{11}	Num	$\longrightarrow$	Any numeric literal (drawn from $\mathcal{R}$)

L_7 An expression Expr evaluates to the logical true or false constant or an inequality involving Attr and Num.

L_6 A condition Cond can be an expression (Expr) or a conjunction or disjunction of expressions.

L_5 An action m consists of condition Cond and two lists (as discussed later, these are the DeleteList and AddList, respectively). The elements of each list are comma-separated. An example of an action is m(true, {PPE, pollution}, {PPE+=[50, 100], pollution+=PPE∗[0.2, 0.4]}). Here the condition is the logical constant true. The DeleteList consists of the prior values of PPE and pollution. The AddList consists of the updated values of PPE and pollution. For example, let's assume the current state s has a PPE value of 10. After executing the action, the new value of the state variable PPE can be updated with any value in the range [50, 100], plus the previous value. If we consider a PPE value of 60 from the range, the next PPE value in the next state s' would be 70.

L_3-L_4 A commitment consists of two agents (debtor and creditor) and a pair of conditions, with the debtor promising the creditor that it will make the second condition true if the first condition is made true. Similarly, a prohibition involves the debtor agent promising the creditor agent to not let the second condition become true if the first condition is made true. Consider the following example prohibition: p(CompanyNear, Regulator, true, pollution $\geq$ 60). Here, the debtor is CompanyNear and the creditor is Regulator. CompanyNear promises Regulator that it will always (the first condition is always true) ensure that the second condition does not become true (pollution exceeding 60ppm).

L_2 A norm may be a prohibition or a commitment.

L_1 An STS specification consists of one or more norms and one or more actions.

We will use Listing 1 to explain the technical and social layers of an STS specification in the following sections.

3.2 Social layer: norms

Following Kafalı et al. [27] and Singh [51], we define a *norm* as a tuple $\langle n, \text{SBJ}, \text{OBJ}, \text{ant}, \text{con}\rangle$, where n represents the norm type from {c, p}; SBJ $\in \mathbb{AG}$ is its subject; OBJ $\in \mathbb{AG}$ is its object; ant and con are conditions (Cond in Table 1 above) that represent the antecedent and consequent of the norm, respectively. The set of norm types $\{c, p\}$ consists of *commitments* (c) and *prohibitions* (p).

A commitment indicates that the subject is committed to its object to making the consequent true if the antecedent holds. Consider the following commitment from Listing 1: C_1(CompanyNear, Hospital, true, PPE $\geq$ 100). The CompanyNear is committed to the Hospital to always (note that the antecedent is true) producing 100 units of PPE or more. The company is the "debtor" for this commitment, and it would violate its commitment if it fails to produce the specified amount of PPE.

A prohibition (p) indicates that its subject is prohibited by its object from making the consequent true when the antecedent holds. Consider the following prohibition from Listing 1: P_1(CompanyNear, Regulator, true, pollution $\geq$ 60). For example, the regulator prohibits the company from polluting above 60 ppm at any time. CompanyNear would violate its prohibition if pollution goes above the specified level of the chemical in question.

As seen in Listing 1, we have used a subscript number with each norm type, where the subscript denotes the instance of each norm type (e.g., C_1 and C_2 are two instances of commitments). For example, in Listing 1 each company has one commitment, so we have defined commitment C_1(CompanyNear, Hospital, true, PPE $\geq$ 100) for CompanyNear and C_2(CompanyFar, Hospital, true, PPE$\geq$100) for CompanyFar. Similarly, each company has one prohibition, so we have defined P_1(CompanyNear, Regulator, true, pollution$\geq$60) for CompanyNear and P_2(CompanyFar, Regulator, true, pollution$\geq$80) for CompanyFar.

3.3 Technical layer: actions and state transitions

The technical layer consists of a finite set of operational actions $A = \{a_1, a_2, \ldots, a_k\}$ that agents can execute. The technical layer consists of a finite set of operational actions $A = \{a_1, a_2, \ldots, a_k\}$ for each agent to execute. In our use case, CompanyNear can choose to manufacture 100 PPE units ($a11$) or 120 PPE units ($a12$) and similarly, CompanyFar can choose to manufacture 100 PPE units ($a21$) or 120 PPE units ($a22$).

An action is represented as m(condition, DeleteList, AddList), where condition is a Cond (in the sense of Table 1) while each of AddList and DeleteList is a List (also in the sense of Table 1). If an action is applicable (i.e., the condition holds in the current state), the action is executed, leading to a transition to a new state where the old values of the attributes contained in DeleteList are removed and the new values of the attributes specified in AddList are added. The outcome of an action can be either deterministic or nondeterministic. The AddList can specify deterministic or nondeterministic outcomes. The use of a range indicates that the outcome of the execution of that action can lead to a state where the variable in question is assigned any value within its range. In Listing 1 below, the execution of the action $a11$ can lead to a state where the new value of the state variable PPE can be any value in the range [50,100] and the new value of the state variable $pollution$ is the value of PPE multiplied by any step-size in the real interval [0.2, 0.4].

3.3.1 Selection probability

The ReNo should support agent autonomy; agents are free to select which action they wish to execute, but this choice is informed by the applicable norms. We would like to select an action which has the most compliant behaviour. We consider how likely an action is compliant to each norm. Because an action may be more compliant to a particular norm but at same time,

Listing 1 STS specification for PPE manufacturing.

```
P1(CompanyNear, Regulator, true, pollution≥60)
C1(CompanyNear, Hospital, true, PPE≥100)
P2(CompanyFar, Regulator, true, pollution≥80)
C2(CompanyFar, Hospital, true, PPE≥100)

a11: m(true, {PPE, pollution}, {PPE+=[50, 100],
   pollution+=PPE*[0.2, 0.4]})
a12: m(true, {PPE, pollution}, {PPE+=[80, 120],
   pollution+=PPE*[0.5, 0.7]})
a10: m(true, {PPE, pollution}, {PPE+=[0, 0],
   pollution+=PPE*[0.0, 0.0]})

a21: m(true, {PPE, pollution}, {PPE+=[50, 100],
   pollution+=PPE*[0.2, 0.4]})
a22: m(true, {PPE, pollution}, {PPE+=[80, 120],
   pollution+=PPE*[0.5, 0.7]})
a20: m(true, {PPE, pollution}, {PPE+=[0, 0],
   pollution+=PPE*[0.0, 0.0]})
```

not compliant to any other norms. Our framework ReNo helps the agent to select an action from a set of actions that has the most compliant behaviour. For instance, in a given state, CompanyFar can execute either a21 and a22 to manufacture 100 PPE. In a given state, let's assume that there is a 100% chance of action a21 complying with a commitment, while there is a 10% chance of action a21 complying with a prohibition. Similarly, let's assume that there is a 50% chance of action a22 complying with a commitment, while there is an 80% chance of action a21 complying with a prohibition. In this scenario, the ReNo will select action a21, which demonstrates the highest level of compliant behavior by considering both commitment and prohibition for execution in that particular state.

Therefore, we wish to model both agent choice (i.e., which action to execute) and transition probabilities (uncertain outcomes accruing from the execution of an action). We make a distinction between *selection probability* and *execution (or transition) probability*. The selection probability is important because we use *selection probability* to determine the norm-compliant behaviour.

We use the common mathematical form "softmax" to map weights to probabilities. Softmax or normalized exponential function is a mathematical function used to convert a vector of $\mathcal{R}$ real numbers into a probability distribution of $\mathcal{R}$ possible outcomes. The softmax function normalizes the input vector, making sure that the resulting values range between 0 and 1 and that their sum adds up to 1, which is a requirement for a probability distribution. Consequently, the output vector obtained from softmax can be interpreted as a probability distribution over the different classes or categories. The essential intuition is that selection is a means to deal with autonomy while execution probabilities are a way to deal with a stochastic environment. Here, a stochastic environment means one in which instructing an agent to take a particular action does not necessarily lead to that action being carried out.

We define a function $prob(a_k, s, A, N)$ (Eq. (1)) which calculates the likelihood of an agent choosing an action from a given state (i.e., selection probability). This function computes the propensity of an agent selecting an action for execution, given the impact of one execution of the action on compliance with the applicable set of norms. When propensity is computed for all actions, we transform the resulting probabilities into a probability distribution for the set of actions.

$$prob(a_k, s, A, N) = \prod_{i=1}^{|N|} \frac{w_i^{sat(a_k,n_i)}}{\sum_{a \in A} w_i^{sat(a,n_i)}} \tag{1}$$

where $prob(a_k, s, A, N)$ indicates the probability of selecting action a_k from a set of actions A in a state s, $sat(a_k, n_i)$ indicates the probability that action a_k satisfies the current target set out by norm n_i, and w is a parameter that indicates our willingness to permit the norms that instantiate n_i to be violated (e.g., a higher w indicates that a norm violating action is less likely to be picked). The weight w_i is not a part of the norm n_i. The willingness to violate the norm could be contingent on the agent, the current state or the potential norm-violating action, or all three. We do not propose to be prescriptive here about which intuition needs to be adopted. Indeed, any of these intuitions might be valid. We simply provide machinery in this equation to obtain a selection probability based on the willingness to violate the norm.

Now we will describe the process of computing $sat(a_k, n_i)$. The Eq. (2) is used to capture the likelihood of an action being compliant with prohibitions. Equation (3) captures the likelihood of an action being compliant with commitments.

For an action a_k, a norm n_i, and an attribute Attr that is common to a_k and n_i, we set the probability of a_k satisfying n_i to 0 if the upper-bound for Attr as described in a_k is less than the current target for Attr as described in n_i (as for a commitment) or the lower-bound for Attr as described in a_k is greater than or equal to the current target for Attr as described in n_i (as for a prohibition). Similarly, we set the probability of a_k satisfying n_i to 1 if the lower-bound for Attr as described in a_k is greater than or equal to the current target for Attr as described in n_i (commitment) or the upper bound for Attr as described in a_k is less than the current target for Attr as described in n_i (prohibition). Note that by computing $sat(a_k, n_i)$ as a probability, we eliminate cases where (unnecessarily) increasing or decreasing the amount of a given attribute beyond the target value might have an undesired effect on calculating the likelihood of a given action. For example, the probability of satisfaction is the same if the action produces exactly the target value or twice the target value. If the current target for Attr as described in n_i is within the interval for Attr as described in a_k, then we compute the probability of satisfaction based on a uniform distribution of values from the interval for Attr as described in a_k. For example, if 40% of the values for Attr within the lower bound and upper bound satisfies the current target, then $sat(a_k, n_i)$ is 0.4.

$$c\_target(\text{Attr}) = n_i\_target(\text{Attr}) - current(\text{Attr})$$

$$sat(a_k, n_i) = \begin{cases} 1, & \text{if } u_{\text{Attr}} \leq c\_target(\text{Attr}) \\ 0, & \text{if } l_{\text{Attr}} \geq c\_target(\text{Attr}) \\ \dfrac{c\_target(\text{Attr}) - l_{\text{Attr}}}{u_{\text{Attr}} - l_{\text{Attr}}}, & \text{otherwise} \end{cases} \tag{2}$$

$$sat(a_k, n_i) = \begin{cases} 1, & \text{if } l_{\text{Attr}} \geq c\_target(\text{Attr}) \\ 0, & \text{if } u_{\text{Attr}} \leq c\_target(\text{Attr}) \\ \dfrac{u_{\text{Attr}} - c\_target(\text{Attr})}{u_{\text{Attr}} - l_{\text{Attr}}}, & \text{otherwise} \end{cases} \tag{3}$$

where n_i_target(Attr) is the target value for the consequent of norm n_i, current(Attr) is the value of the attribute in the current state, l_{Attr} and u_{Attr} are lower bound and upper bound of attribute Attr in the DeleteList or AddList of a_k.

Now, we provide a proof for the Eq. (1) that it provides a probability distribution for the set of actions in a given state s.

Theorem 1 *To prove that the sum of probabilities of all actions for state s is equal to 1, we need to sum the probabilities of all actions over the set of available actions A:* $\sum_{a_k \in A} prob(a_k, s, A, N) = 1$

Proof Let's substitute the value of $\sum_{a_k \in A} prob(a_k, s, A, N)$ from Eq. (1).

$$\sum_{a_k \in A} prob(a_k, s, A, N) = \sum_{a_k \in A} \prod_{i=1}^{|N|} \frac{w_i^{sat(a_k, n_i)}}{\sum_{a \in A} w_i^{sat(a, n_i)}} = \prod_{i=1}^{|N|} \sum_{a_k \in A} \frac{w_i^{sat(a_k, n_i)}}{\sum_{a \in A} w_i^{sat(a, n_i)}} \tag{A}$$

$$\prod_{i=1}^{|N|} \sum_{a_k \in A} \frac{w_i^{sat(a_k, n_i)}}{\sum_{a \in A} w_i^{sat(a, n_i)}} = \prod_{i=1}^{|N|} \frac{\sum_{a_k \in A} w_i^{sat(a_k, n_i)}}{\sum_{a \in A} w_i^{sat(a, n_i)}} \tag{B}$$

Now let's take $\frac{\sum_{a_k \in A} w_i^{sat(a_k, n_i)}}{\sum_{a \in A} w_i^{sat(a, n_i)}}$ from Eq. (B) and using the properties of summation, we can split the sum into two parts.

$$\frac{\sum_{a_k \in A} w_i^{sat(a_k, n_i)}}{\sum_{a \in A} w_i^{sat(a, n_i)}} = \frac{w_i^{sat(a_1, n_i)}}{\sum_{a \in A} w_i^{sat(a, n_i)}} + \frac{w_i^{sat(a_2, n_i)}}{\sum_{a \in A} w_i^{sat(a, n_i)}} + ... + \frac{w_i^{sat(a_n, n_i)}}{\sum_{a \in A} w_i^{sat(a, n_i)}}$$

Now, let's bring the common denominator, $\sum_{a \in A} w_i^{sat(a, n_i)}$, inside each term:

$$\frac{w_i^{sat(a_1, n_i)} + w_i^{sat(a_2, n_i)} + ... + w_i^{sat(a_n, n_i)}}{\sum_{a \in A} w_i^{sat(a, n_i)}}$$

Since the denominator$\sum_{a \in A} w_i^{sat(a, n_i)}$ is the sum of the exponential values of all actions in the A so it can be canceled out.

$$\frac{w_i^{sat(a_1, n_i)} + w_i^{sat(a_2, n_i)} + ... + w_i^{sat(a_n, n_i)}}{\sum_{a \in A} w_i^{sat(a, n_i)}} = 1$$

So,

$$\frac{\sum_{a_k \in A} w_i^{sat(a_k, n_i)}}{\sum_{a \in A} w_i^{sat(a, n_i)}} = 1 \tag{C}$$

Now put value of $\frac{\sum_{a_k \in A} w_i^{sat(a_k, n_i)}}{\sum_{a \in A} w_i^{sat(a, n_i)}}$ from Eq. (C) in Eq. (B).

$$\prod_{i=1}^{|N|} \frac{\sum_{a_k \in A} w_i^{sat(a_k, n_i)}}{\sum_{a \in A} w_i^{sat(a, n_i)}} = \prod_{i=1}^{|N|} *1 = 1$$

Hence,

$$\sum_{a_k \in A} prob(a_k, s, A, N) = 1$$

□

3.3.2 Translation of an STS specification to a PRISM model

The PRISM model is constructed as a probabilistic state transition model. In the following algorithms, c_target(Attr), n_target(Attr), and current(Attr) have the same denotations as in Eqs. (1, 2, and 3). In Algorithm 1, we compute a PRISM model that includes a transition probability for every feasible transition $\langle s, s' \rangle$ where s' is one of the states that could result from executing action a in state s. We also note that although our state schema includes a variable that denotes the agent whose turn to act, this is entirely an artifact of the round-robin agent execution technique we are using for approximately checking an STS specification. In the algorithms defined below, we can ignore the state variable called *agent* except in situations where we explicitly refer to it.

We also note that for many of the actions we specify their resulting states in terms of the deltas (i.e., the operations performed) on the prior values of the state variables. In the algorithms below, we avoid this complexity by simply assuming that the actions are specified in terms of the values of the state variables in the resulting state instead of explicitly describing the changes or operations applied to the state variables. Thus, the algorithms avoid the complexity of detailing how the state variables are updated. They focus on the end result, rather than the specific steps involved in achieving that result.

Algorithm 1 translates an STS specification into a PRISM model. The algorithm takes two inputs: (1) An STS specification consisting of a set of agents, where each agent has a set of norms N and a set of actions A. (2) The set of action execution probabilities given by ACTIONEXECPROB (these are the probabilities associated with the action execution transitions). In practical settings, we expect these probabilities to be obtained from past execution data (for this paper's experimental purposes, these values are randomly generated).

- We create the initial state s_0 by assigning 1 to variable *agent* and assigning random values to other state variables (Line 2).
- The consequences of actions always lie in that range. The number of outcomes of each action is calculated based on a step size. If we use a small value for step-size, then we have more number of outcomes of an action as compared to a large value for step-size. For example, the consequences of the action $a11$ always lie between 50 and 100. If the STS designer chooses the step size as 10 then this action has six consequences, i.e., 50, 60, 70, 80, 90, and 100 whereas if the step size is 20 then it has only three consequences, i.e., 50, 70, and 90. It can be noticed that action $a11$ has six consequences when step-size is 10 whereas action $a11$ has three consequences when step-size is 20.
- In Line 5, when a specific agent is selected in a state then all transitions from this state lead to the states where we execute the set of actions (determined by the non-zero value of an agent, which serves as an identifier for the selected agent). Then, we add transitions corresponding to all non-deterministic outcomes of executions of the actions identified by the value of the *agent* variable (Line 6 or Line 9).
- We build a PRISM model as a probabilistic state transition model. Hence, we need to compute the transition probability for each state transition. Each state's state transition probability is calculated by multiplying the selection probability of an action we execute with the associated probabilities for that action as defined by the input ACTIONEXECPROB (Algorithm 1). For a state where a specific agent is selected, we calculate the action selection probability for each action in the set of actions A for that agent using Eq. (1).
- Algorithm 2 is used to calculate an action selection probability in a given state.
  - We go through each action a' in the set of actions A (Line 2 in Algorithm 2).

Algorithm 1 Generating PRISM model from an STS specification.

Input: An STS specification consisting of a set of agents, where each agent is specified via:
A set of norms N
A set of actions A
Input: w is a user defined parameter that indicates our willingness to permit the norms that instantiate n where $n \in N$ to be violated.
Input: A function ActionExecProb : $A \times S \rightarrow \mathbb{R}$ which is the probability of arriving in a given state in S by executing an action in A, independent of which state we are in when this action is executed.
Output: A PRISM model $(S, A, TransitionProb)$ where S is a set of states, A is a set of actions and a function $TransitionProb : S \times S \rightarrow \mathbb{R}$

```
1: Create an initial state s_0 by assigning to $agent$ the ID of the first agent in the input STS specification and
   assigning the value 0 to all the other state variables (note that all of these are necessarily numeric valued).
2: S:= {s_0}
3: while S ≠ ∅ do
4:     Let s ∈ S
5:     for each $a \in A$ for the agent denoted by the current value of $agent$ do
6:         if $agent$ ≠ last agent in the STS specification then
7:             s′ is obtained from s by removing value assignments to variables that appear in the
               DeleteList of $a$ and adding variable value pairs that appear in the AddList of $a$;
8:             $agent$ := next agent in the STS specification.
9:         else if $agent$=last agent in the STS specification then
10:            s′ is obtained from s by removing value assignments to variables that appear in the
               DeleteList of $a$ and adding variable value pairs that appear in the AddList of $a$;
11:            $agent$ := first agent in the STS specification.
12:        end if
13:        $TransitionProb(s, s')$ :=
           ActionSelectionProbability(s,$w$,$a$,$A$,$N$)
           *ActionExecProb($a$, $s'$)
14:    end for
15:    S:= S-{s}
16: end while
```

- Then for each action a', we go through each norm n in the set of norms N to compute action a''s probability which satisfies the current target set out by the norm n (Line 3 in Algorithm 2).
- If norm n is prohibition then we use line 4 to line 7 in Algorithm 2 to compute action a''s probability which satisfies the current target set out by the norm prohibition.
- Similarly, if norm n is commitment then line 8 to line 11 in Algorithm 2 are used to compute the actions a' that satisfies the current target set out by the norm commitment. For example, agent CompanyNear is selected in the current state s. Then, we calculate the action selection probability for each action $a11$, $a12$, and $a10$ based on the norms P_1 and C_1 (from Listing 1).
- Finally, we get the action selection probability for each action using Eq. (1) (Line 14 in Algorithm 2).

- Finally, Line 13 in Algorithm 1 will be used to compute the transition probability for the updated state which is added using either Line 7 or Line 10 in Algorithm 1.
- For example, the current agent is CompanyNear in state s. For each action, such as $a11$, the state variables are updated (which results in the current state being updated to the next state) and state transition probability for the next state is computed in Line 13 in Algorithm 1.
- Line 13 computes the state transition probability which is equal to the multiplication of the selection probability of $a11$ with its action execution probability. This process is repeated for the remaining actions, i.e., $a12$ and $a10$. As discussed in Sect. 3.3, the

states resulting from these transitions are obtained from the corresponding prior states by removing the value assignments to variables in the DeleteList and adding the new values to these variables as specified in the AddList of the action.

Algorithm 2 Calculating selection probability in a given state s.

```
1: function ACTIONSELECTIONPROBABILITY(s,$a$,$w$,$A$,$N$) where $a$ is the action of interest (i.e., the action
   whose probability of selection we wish to compute), $s$ is the current state and $N$ is the currently applicable
   set of norms
2:     for each $a' \in A$ for the applicable agent in s do
3:         for each $n \in N$ for the applicable agent in s do
4:             if $n$ is a prohibition and state variable Attr is common to $a$ and $n$ then
5:                 c_target(Attr) := n_target(Attr) − current(Attr)
6:                 $sat(a', n)$ is assigned the value computed using Eq. (2)
7:             end if
8:             if $n$ is a commitment and state variable Attr is common to $a$ and $n$ then
9:                 c_target(Attr) := n_target(Attr) −current(Attr)
10:                $sat(a', n)$ is assigned the value computed using Eq. (3)
11:            end if
12:        end for
13:    end for
14:    ACTIONSELPROB(s,$a$, $A$,$N$):= action selection probability computed using Eq. (1).
15:    return ACTIONSELPROB(s, $a$,$A$,$N$)
16: end function
```

Note that we assume that each action and each norm refers to only one state variable (i.e., only one Attr).

Example of PRISM model generated using Algorithm 1. It would be easier to visualise the model graphically. In PRISM, the transition matrix of the model can be exported in the Dot (Dot is a graph description language) format, which allows easy graphical visualisation of the graph structure of the model. Figure 2 is constructed based on the transition matrix of a PRISM model generated using Algorithm 1 from a simplified STS specification with two agents where each agent has two actions. Figure 2 shows part of a PRISM model but the complete diagram is provided in the supplementary material. We describe the application of our algorithm from State 31 (2,3,1)—the red box in the diagram, where agent = 2, PPE = 3, and pollution = 1. Since the agent = 2, the CompanyNear agent is selected as a current agent. Two actions are executed next and each action has two consequences. In State 31, there are four potential state transitions: with 0.11 probability PPE is increased by one in the next state (State 1—green box) where agent =1; with 0.235 probability PPE is increased by four and pollution is increased by one where agent = 1 in the next state (State 3 green box); with 0.265 probability PPE is increased by four and pollution is increased by two where agent = 1 in the next state (State 4 green box). In state 4 (1,7,3)—the green box in the diagram where agent = 1; PPE = 7; and pollution = 3. Since the agent = 1, the CompanyFar agent is selected as a current agent. There are two actions for CompanyFar to be executed next. There are four resulting states, such as 37 (2,8,3) and 54 (2,14,5). Then, state 37 (2,8,3) delivers agent = 2 so CompanyFar is again selected using the round-robin agent execution technique.

3.4 Requirements as PRISM properties

ReNo supports three core types of requirements: achievement and maintenance requirements are essential to the norm literature (e.g., a commitment to achieve something to a certain

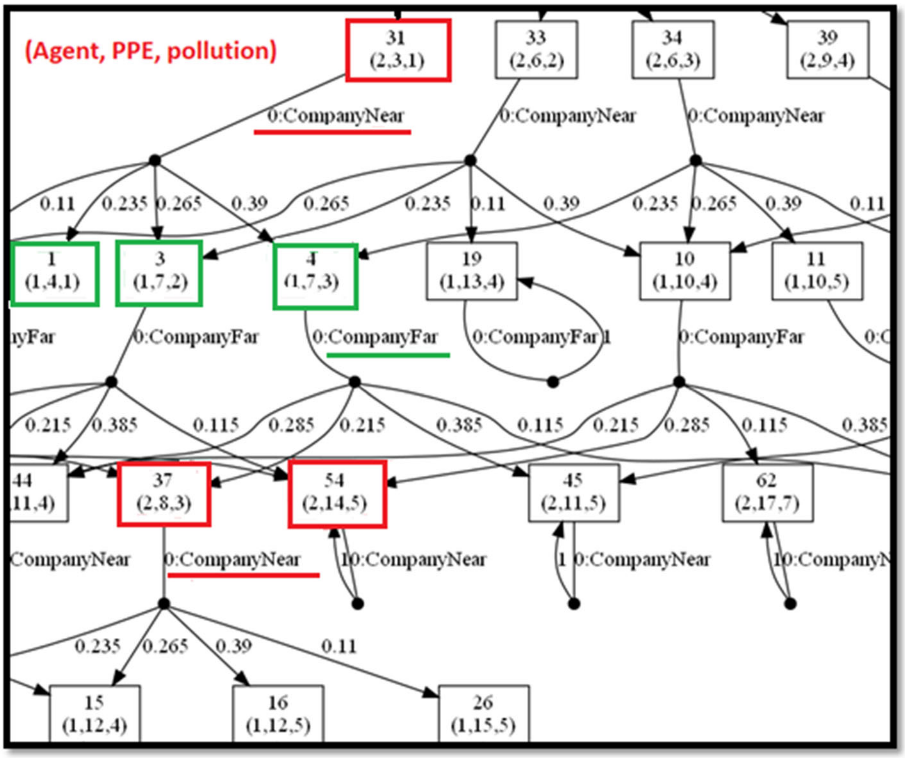

Fig. 2 Sample (partial) PRISM model using Algorithm 1

level or a prohibition to maintain something at a certain level). The third type is the resilience requirements (novel to RENO) to verify that an STS can recover from requirement failures. We believe these three types of requirements cover realistic verification scenarios, to help guide STS designers. In the following, we consider these three types of requirements and their formalization in PRISM syntax.

Achievement and Maintenance. Consider the requirement that the probability that the company can manufacture at least X PPE units within K timesteps and maintain the pollution level below Y ppm should be no less than P. We formalize this in PRISM as follows:

$$P_{min=?}[(pollution < Y) \ \ \mathsf{U}^{\leq K} \ \ (PPE \geq X)]$$

If we achieve minimum probability P ($P_{min=?}$ =P), then this requirement is satisfied.
Resilience Consider a situation where we find the pollution level above some threshold (say X ppm). Then we wish to achieve a better state (where the over-pollution problem has been resolved by bringing it down to below Y ppm) within K steps to be no worse than some threshold value.
We formalize this requirement in PRISM as follows:

$$filter(print, P_{min=?}[\mathsf{F}^{\leq K} pollution \leq Y], pollution \geq X)$$

$filter()$ is a PRISM device that allows us to perform max or min over multiple states satisfying some property (here that property is pollution $\geq$ X). If we achieve a min of 0, that tells us that we are not resilient at all while if we achieve a minimum of 1.0, then we are fully resilient.
For resilience, we are interested in ensuring that our system remains within the states that satisfy the above property with a minimum probability of 0.75. We formalize the second part of the requirement in PRISM as follows:

$$P_{min=?}[\mathsf{G}^{\leq N}(pollution \geq Y \Rightarrow P_{\geq 1}[\mathsf{F}^{\leq K}(pollution \leq X)])]$$

4 Methodological guidelines

In this section, we provide methodological guidance to support the design and implementation of resilient STSs. We provide guidance from two perspectives:

- From the perspective of **STS designers**. Here, the challenge is to ensure that the STS meets the specified requirements while preserving agent autonomy.
- From the perspective of **stakeholders** specifying requirements for the overall system. Here, the challenge is to ensure that the requirements are *feasible*, i.e., that it is not possible to design an STS that satisfies the given requirements.

These are deliberately framed as *methodological guidelines* as opposed to a procedure because there is significant human input and judgment involved. There are multiple options available to the **STS designer** in terms of how the constituent agents (and in particular available agent actions) are designed and in terms of the kinds of norms that should be specified. There are, similarly, multiple options available to stakeholders in terms of how they might design requirements.

4.1 The STS designer perspective

The steps below serve as a guide for the STS designer to develop an initial version of the STS specification and conduct an analysis to ensure stakeholder requirements are satisfied. If the stated stakeholder requirements are not satisfied, the STS designer can iterate the specification to refine it and ensure that stakeholder requirements are satisfied.

Step 1: Defining the agents constituting the STS. The STS designer needs to define each constituent agent as a collection of actions, based on the known capabilities of each agent.

Step 2: Defining the augmented probabilistic state transition model. In general, a probabilistic state transition model describes the likelihood of achieving a resultant state if a specific action is performed in a given state. These probabilities are either provided as input (in the case of *transition probability*) or computed from an agent's willingness to violate a norm (the parameter w which is used in computing the *selection probability* for an action). In this step, the **STS designer** needs to create an augmented probabilistic state transition model that serves as the model for model-checking PRISM based on the two sets of probabilities: *selection probabilities* and *transition probabilities*.

Step 3: Defining the norms governing the STS. There are some common principles guiding the specification of the initial set of norms given a set of available agents and a set of requirements. A resilience requirement specifies two kinds of conditions: (1) Conditions that flag a departure from the "normal" operating mode and (2) Conditions that flag the restoration of the normal operating mode. The overall requirement is augmented with a deadline (in terms of the number of steps) within which normal operations are restored, and a lower bound on the probability that this will happen. The normal operating mode can be achieved via a combination of commitments and prohibitions (e.g., *CompanyNear* committing to *Hospital* to produce at least 100 PPE units per day, but being prohibited by *Regulator* from polluting at a rate higher than 60 parts per million per day). The deviation from the normal operating mode is flagged via a condition where the minimum PPE production target is not met and the maximum permitted pollution level is exceeded. Achievement and maintenance requirements can similarly be specified via combinations of commitments and prohibitions.

Step 4: Using PRISM to determine requirements satisfaction. Our technique enables the encoding of requirements in the form of PCTL properties that can be provided to PRISM as input, and an augmented probabilistic state transition model. When we run PRISM with these inputs, we obtain as output an indication of whether the PCTL properties (and hence the original requirements) are satisfied. If PRISM indicates that these properties have not been satisfied, it triggers a redesign of the STS.

Step 5: Redesigning the STS in light of model checking results. Redesigning the STS can involve (1) Adding or removing agent actions and (2) Adding or removing norms. Adding an action, if feasible, is one approach to resolving a case of requirements failure. If the analysis using PRISM reveals that an achievement goal/requirement is not being satisfied, a new action (for instance to produce PPE at a faster rate) can serve as a potential (re-design) solution. A violation of a maximum permissible pollution requirement can similarly be resolved by disallowing/removing an action that permits a faster rate of PPE production, which comes with a concomitant higher rate of pollution. Adding a new norm in the form of a commitment to produce higher levels of PPE can help resolve a situation where an achievement goal/requirement involving meeting a production target for PPE is not being satisfied.

4.2 The stakeholder perspective

The stakeholder may be required to engage in an iterative process of progressively relaxing the applicable requirements if the STS designer determines that revising the STS specification (Step 5 in Sect. 4.1) cannot meet the requirements.

Modification of the applicable requirements can involve three kinds of changes:

- Modifying the associated conditions: In our examples, these mainly involve inequalities on state variables (such as the number of PPE units, or the pollution level). These are entirely left to human judgment.
- Modifying the number of steps (i.e., the deadline) and the minimum associated probability: Both of these can be conjointly modified and represent a trade-off space for the stakeholder. Consider, for instance, Fig. 3 which shows the minimum probabilities achievable for a given number of steps. Figure 6 similarly illustrates the trade-off space in the context of a resilience requirement. Accessing a visualization of the trade-off space will make it easier for a stakeholder to decide on an appropriate relaxation of a requirement.

5 Prototype implementation and requirement verification results

This section describes the evaluation we carried out, for our approach. We describe the requirement verification results of our use case. Then, we discuss our experimental settings, scalability of the implementation, performance measures, and finally, report our results.

To construct and analyse a model with PRISM, it must be specified in the PRISM language, a simple, state-based language. Specifically, the PRISM language is composed of modules and variables. The values of these variables at any given time constitute the state of the module. The behavior of the module is described by a set of commands.

$$[action]guard \Rightarrow prob_1 : update_1 + \ldots + prob_n : update_n$$

The **guard** is a predicate over all the variables in the model. Each **update** describes a transition that can be made by a module where the guard is true. A transition relationship of the system is specified by allocating new values to the variables in the module. Each update is also assigned a **probability** (or in some cases a rate) - assigned to the corresponding transition. If a module has more than one variable, then updates describe the new value for each one of them. The initial state of a model is then defined by the initial value of all variables (see Appendix A.1) for the initial state in the PRISM model). However, the PRISM input language does not support external functions, such as function $prob(a_k, s, A, N)$ in Eq. (1), to dynamically construct a state-transition model. Therefore, we have connected to PRISM programmatically to create a model building on the machinery described in Eq. (1) (see Appendix A.2).

PRISM provides a Java-based interface or API that allows users to interact with PRISM programmatically. This enables us to automate the construction, solving, and verification of the PRISM model. Algorithm 1 has been used to automatically construct the PRISM model by translating an STS specification into a PRISM model that incorporates dynamic state transitions. That is, state transition probabilities are recalculated after each transition depending on the values of the state variables. We show the snippets for state transitions in the PRISM model in Appendix A.3.

The first step in our framework in Fig. 1 is to capture the STS specification. Here, Listing 1 describes the STS specification for PPE manufacturing with two companies. Each company has three actions where two actions are used to manufacture PPE and one action describes a *null* action. We introduce a null action where the company does not want to manufacture PPE. Specifically, actions $a10$ and $a20$ are considered as null actions for CompanyNear and CompanyFar, respectively. The second step in our framework, stakeholder provide a set of requirements. So we use the requirements introduced in Sect. 3.4. Then, the STS specification is automatically converted into a PRISM model using Algorithm 1. Finally, we demonstrate the verification results for the requirements introduced in Sect. 3.4 to see how likely STS satisfies or fails each requirement. Section 5.1 and Sect. 5.3 have been used to show that if a requirement is not satisfied then the STS designer can revise the STS specification. Section 5.2 and Sect. 5.4 have been used to show that if a requirement is not possible to satisfy then the stakeholders can relax the requirement to satisfy with a given STS.

Tables 2 and 3 illustrate the probability needed to satisfy a given requirement with a given STS specification. If a requirement is noncompliant with an STS specification, then the STS specification needs to be refined to satisfy a given requirement. Tables 2 and 3 also capture the results where we design the new requirements by relaxing the value of state variables based on results from the analysis phase. The results shown in Tables 2 and 3 have been derived by using methodological guidelines mentioned in Sect. 4.

Table 2 Experimental settings for achievement and maintenance

Requirement	Case being verified	STS	Outcome
Eq. (4)	Original (Case 1)	Listing 1	Not satisfied even if the requirement is relaxed
	CompanyNear is productive (Case 2)	Listing 2	Satisfied if relaxed to 8 steps
	Both are productive (Case 3)	Listing 3	Satisfied as stated
Eq. (5)	Original	Listing 1	Satisfied as stated

We present various cases, each corresponding to an STS specification, and verify whether the specification satisfies the requirement. In the paper, we have provided listings to show an STS specification for each case. However, the data such as outcomes of actions and execution probabilities used to conduct experiments is provided in the supplementary material [34].

5.1 Handling strict achievement and maintenance requirements

The requirement below is an instance of the achievement and maintenance requirement discussed above. The minimum probability that the company manufactures more than 200 PPE units within six timesteps and maintains the pollution level below the threshold of 70 ppm within six timesteps is 0.75. If we achieve a minimum probability of 0.75, then this requirement is satisfied. Table 2 shows the experimental settings for achievement and maintenance requirements. Each case in Table 2 corresponds to an STS specification.

We verify whether a case satisfies the requirement.

$$P_{min=?}[(pollution \leq 70) \quad \mathsf{U}^{\leq 6} \quad (PPE \geq 200)] \tag{4}$$

Case 1 The STS in this case follows the specification of Listing 1. This requirement is only satisfied if both companies produce more than 200 PPE within six timesteps with a minimum probability of 0.75. The minimum probability that both companies produce more than 200 PPE within six timesteps is 0.69, hence, this requirement is not satisfied.

Figure 3 illustrates that the minimum probability to be satisfied for this property varies with the number of timesteps (k). Moreover, 0.73 is the probability achieved after increasing the k number of time steps to ten. Hence, this requirement cannot be satisfied within the timesteps allowed.

Case 2 The STS is revised for CompanyNear. The Listing 2 shows a revised specification of Listing 1 – commitment C_1 is replaced by C$_3$(CompanyNear, Hospital, true, PPE≥200). After replacing commitment C_1 by C_3, CompanyNear is more productive in terms of manufacturing more PPEs. Commitment C_3 enables the selection of an action for increasing the production of PPEs. The minimum probability achieved is 0.69 (achieved in six timesteps). Hence, this requirement is not satisfied. Figure 3 shows that the requirement is satisfied within eight timesteps with 0.77 minimum probability.

Case 3 As we have seen in Case 2 this requirement is satisfied in eight or more timesteps and cannot be satisfied within six timesteps, hence, we revise STS for CompanyFar. STS follows Listing 3, a revision of Listing 2—commitment C_2 is replaced by C$_4$(CompanyFar, Hospital, true, PPE≥200). In this case, the commitment to manufacture PPE has been increased for both the companies. Therefore, both companies are productive as they choose the actions that manufacture more PPE. Figure 3 shows the minimum probability achieved is 0.75 (achieved in six timesteps). Hence, this requirement is satisfied.

Listing 2 Revised STS specification of Listing 1 for CompanyNear to ensure achievement and Maintenance.

```
Addition of norms or actions are shown in blue
    and deleted lines are shown with red
    strikethrough.

P1(CompanyNear, Regulator, true, pollution≥60)
~~C1(CompanyNear, Hospital, true, PPE≥100)~~
C3(CompanyNear, Hospital, true, PPE≥200)
P2(CompanyFar, Regulator, true, pollution≥80)
C2(CompanyFar, Hospital, true, PPE≥100)

a11: m(true, {PPE, pollution}, {PPE+=[50, 100],
    pollution+=PPE*[0.2, 0.4]})
a12: m(true, {PPE, pollution}, {PPE+=[80, 120],
    pollution+=PPE*[0.5, 0.7]})
a10: m(true, {PPE, pollution}, {PPE+=[0, 0],
    pollution+=PPE*[0.0, 0.0]})

a21: m(true, {PPE, pollution}, {PPE+=[50, 100],
    pollution+=PPE*[0.2, 0.4]})
a22: m(true, {PPE, pollution}, {PPE+=[80, 120],
    pollution+=PPE*[0.5, 0.7]})
a20: m(true, {PPE, pollution}, {PPE+=[0, 0],
    pollution+=PPE*[0.0, 0.0]})
```

It is interesting to note that the minimum probability of satisfying this requirement is less in Case 2 than in Case 1 with less than six timesteps. Here, CompanyFar is committed to producing more than 200 PPE, which leads it to select action a21 with priority, which results in high pollution in the short term. So, CompanyFar meets the social objective but the probability of choosing action a21 is expensive in terms of increasing pollution.

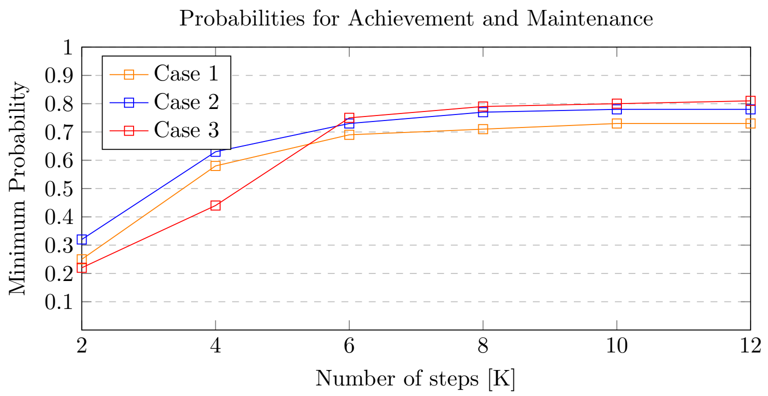

Fig. 3 Probabilities for achievement and maintenance

Listing 3 Revised STS specification of Listing 2 to ensure achievement and maintenance.

```
Addition of norms or actions are shown in blue
    and deleted lines are shown with red
    strikethrough.

P1(CompanyNear, Regulator, true, pollution≥60)
C3(CompanyNear, Hospital, true, PPE≥200)
P2(CompanyFar, Regulator, true, pollution≥80)
~~C2(CompanyFar, Hospital, true, PPE≥100)~~
C4(CompanyFar, Hospital, true, PPE≥200)

a11: m(true, {PPE, pollution}, {PPE+=[50, 100],
    pollution+=PPE*[0.2, 0.4]})
a12: m(true, {PPE, pollution}, {PPE+=[80, 120],
    pollution+=PPE*[0.5, 0.7]})
a10: m(true, {PPE, pollution}, {PPE+=[0, 0],
    pollution+=PPE*[0.0, 0.0]})

a21: m(true, {PPE, pollution}, {PPE+=[50, 100],
    pollution+=PPE*[0.2, 0.4]})
a22: m(true, {PPE, pollution}, {PPE+=[80, 120],
    pollution+=PPE*[0.5, 0.7]})
a20: m(true, {PPE, pollution}, {PPE+=[0, 0],
    pollution+=PPE*[0.0, 0.0]})
```

5.2 Handling relaxable achievement and maintenance requirements

As Fig. 1 (the RENO framework) shows, an STS designer can analyze the various state variables to satisfy the requirements. The STS in this case follows the specification of Listing 1. After revising the STS specification (as mentioned in Case 3), we have seen that the minimum probability of achieving more than 200 PPE and maintaining the pollution below 70 ppm is 0.81. The STS designer may want to determine which values of PPE and pollution can satisfy the above property with probability 1.

Figure 4 and Fig. 5 show these minimum probabilities when PPE and pollution vary for a range of values and the number of steps K. Figure 4 describes the minimum probabilities achieved when pollution is less than 70 ppm and PPE varies from 140 to 200. The result shows that the minimum probability 1 is achieved when we have PPE greater than 160 within 10 timesteps or PPE greater than 140 within eight timesteps.

Figure 5 indicates the minimum probabilities achieved when PPE exceeds 200 ppm and pollution varies from 70 ppm to 90 ppm. The result shows that the minimum probability 1 is achieved only when pollution is less than 90 ppm within 12 timesteps.

These values of PPE and pollution indicate to the STS designer to provide new requirements by relaxing the value of PPE and pollution. The observations from Figs. 4 and 5 provide valuable insights for the STS designer. A minimum probability of 0.97 is achieved when we have PPE greater than 140 within six timesteps. Similarly, when pollution is less than 90 within six timesteps, a minimum probability of 0.92 is achieved. Based on these findings,

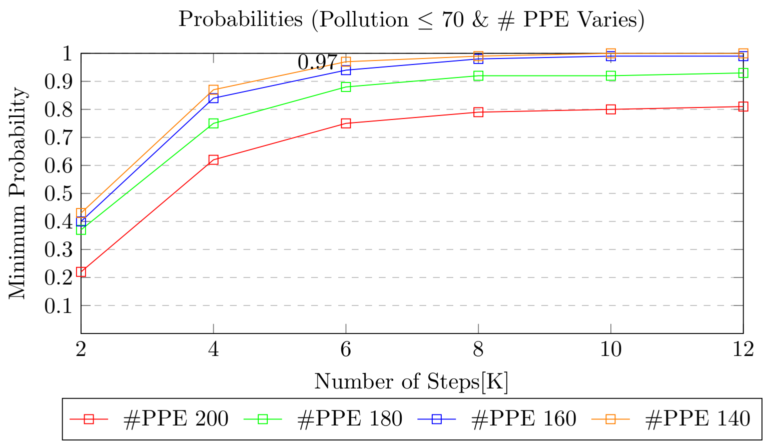

Fig. 4 Probabilities for achievement and maintenance when varying number of PPE units

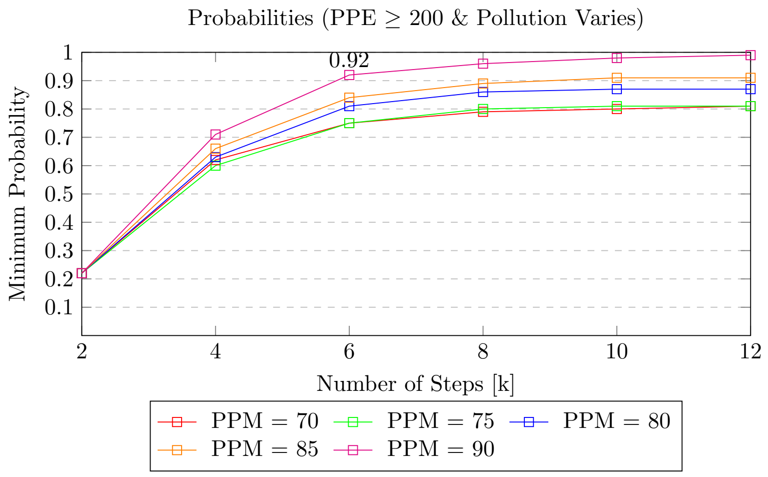

Fig. 5 Probabilities for achievement and maintenance when the varying amount of pollution (PPM)

the STS designer can express the new requirement as follows using these values of PPE and pollution.

$$P_{min=?}[(pollution \leq 90) \quad \mathsf{U}^{\leq 6} \quad (PPE \geq 140)] \tag{5}$$

The minimum probability achieved is 1 (achieved in six timesteps). Hence, the requirement is satisfied with certainty (minimum probability of 1).

5.3 Handling strict resilience requirements

Informally, we view resilience as an indicator of both the speed and likelihood with which a desirable property can be restored when the system of interest is found to be in a state that violates these properties.

This section describes the verification results of the resilience requirement. We aim to verify how the system reacts when pollution exceeds the given threshold value. The threshold

Table 3 Experimental settings for resilience

Requirement	Case being verified	STS	Outcome
Eq. (7)	Original (Case 4)	Listing 1	Not satisfied even if the requirement is relaxed
	Both companies are productive (Case 5)	Listing 4	Satisfied as stated
Eq. (8)	Both are productive (Case 5)	Listing 4	Satisfied as stated

value of the pollution represents a change in the operating environment of the system. Then, lowering the pollution level with probability 1 within a certain number of steps indicates the response to the system. Table 3 shows the experimental setting for resilience. Each case in Table 3 corresponds to an STS specification. We verify whether a case satisfies the requirement.

First, we would like to verify the minimum probability for a system entering a state where pollution is greater than 180 ppm and the system reduces the pollution level to below 100 ppm within five timesteps. This property can be written in PCTL as described in Eq. (6).

$$filter(print, P_{min=?}[\mathsf{F}^{\leq 5} pollution \leq 100], pollution \geq 180) \tag{6}$$

Equation (6) is used to examine the minimum probability required for a state change from a state having pollution greater than 180 ppm to a state having pollution below 100 ppm within five timesteps. After verification, we get 1.0 as the minimum probability for this property.

Second, we are interested in ensuring that our system remains within the states that satisfy the above property (Eq. (6)) for seven timesteps with a minimum probability of 0.75. This property can be written in PCTL as follows:

$$P_{min=?}[\mathsf{G}^{\leq 7}(pollution \geq 180 \Rightarrow P \geq 1[\mathsf{F}^{\leq 5}(pollution \leq 100)])] \tag{7}$$

Equation (7) is a query for the minimum probability of the system staying in states where pollution goes above 180 ppm (threshold), the probability of lowering the pollution below 100 ppm in five timesteps with probability 1 for seven timesteps. Here, the property $P \geq 1[\mathsf{F}^{\leq 5}(pollution \leq 100)]$ represents the response that we are expecting from the system when there is a change in its operating environment ($pollution \geq 180$).

Case 4 The STS in this case follows the specification of Listing 1. The minimum probability achieved for the property of Eq. (7) is 0.43. Hence, this requirement is not satisfied. Figure 6 shows that the minimum probability of recovering the system varies with the number of timesteps. When the number of steps increases, additional units of PPE are produced at the cost of increased pollution. It can be noticed from Fig. 6 that the minimum probability achieved for the property of Eq. (7) is 0.43 in seven timesteps, 0.3 in eight timesteps, and so on. Similarly, when the number of steps is decreased then fewer units of PPE are produced with the benefit of reduced pollution. There may be a possibility of requirement getting satisfied. Hence, we check the minimum probability achieved for the property of Eq. (7) before seven timesteps. It can be noticed that even after reducing the number of steps, the property is not satisfied. For example, once we reduce the number of steps from seven to six and then to five the minimum probabilities achieved are 0.73 and 0.69, respectively. This property is not satisfied because none of the actions of the companies (Listing 1) reduce pollution adequately.

Listing 4 Revised STS specification for CompanyNear and CompanyFar to ensure resilience

```
Addition of norms or actions are shown in blue.
_3

P1(CompanyNear, Regulator, true, pollution≥60)
C1(CompanyNear, Hospital, true, PPE≥100)
C3(CompanyNear, Regulator, pollution≥90,
    pollution≤40)
P2(CompanyFar, Regulator, true, pollution≥80)
C2(CompanyFar, Hospital, true, PPE≥100)
C4(CompanyFar, Regulator, pollution≥90,
    pollution≤40)

a10: m(true, {PPE, pollution}, {PPE+=[0, 0],
    pollution+=PPE*[0.0, 0.0]})
a11: m(true, {PPE, pollution}, {PPE+=[50, 100],
    pollution+=PPE*[0.2, 0.4]})
a12: m(true, {PPE, pollution}, {PPE+=[80, 120],
    pollution+=PPE*[0.5, 0.7]})
a13: m(true,  {PPE, pollution},{PPE+=[50, 100],
    pollution+=PPE*[-0.6, 0.0]})

a20: m(true, {PPE, pollution}, {PPE+=[0, 0],
    pollution+=PPE*[0.0, 0.0]})
a21: m(true, {PPE, pollution}, {PPE+=[50, 100],
    pollution+=PPE*[0.2, 0.4]})
a22: m(true, {PPE, pollution}, {PPE+=[80, 120],
    pollution+=PPE*[0.5, 0.7]})
a23: m(true, {PPE, pollution}, {PPE+=[50, 100],
    pollution+=PPE*[-0.6, 0.0]})
```

Case 5 We revise the STS specification (Listing 4) by adding commitment C_3 and C_4 for both companies. Now, both companies are committed to the Regulator to reduce the pollution level below 40 ppm if pollution increases above 90 ppm during the manufacturing of PPE. The revised STS specification contains actions a13 and a23 for CompanyNear and CompanyFar respectively, to decrease pollution while manufacturing PPE. The STS in this case follows the specification (Listing 4) of Listing 1—commitments C_3 and C_4 are added for both companies.

Figure 6 shows that minimum probability achieved is 0.89 (achieved in seven timesteps). Hence, this requirement is satisfied. As we have seen in Case 4 with the original STS specification (Listing 1) the requirement is not satisfied even after we relax the number of steps but after revising STS specification (Listing 4) the property gets satisfied even though we increase the number of steps from seven to ten with 0.77 probability. After revising the STS specification (Listing 4), both companies meet the social objective of reducing pollution below 40 ppm whereas pollution above 90 ppm triggers actions that are less expensive in terms of environmental pollution. These newly introduced actions of each company manufacture the PPE and do not pollute the environment.

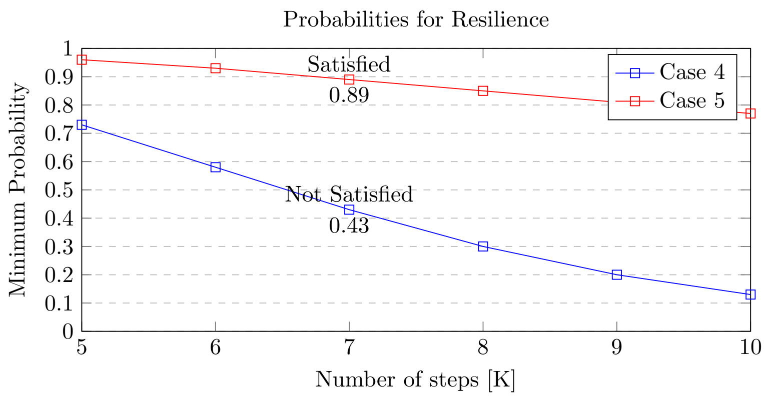

Fig. 6 Probabilities for resilience

5.4 Handling relaxable resilience requirements

We now show how an STS designer can use our framework RENO in Fig. 1 to design the new relaxed requirement. We use both the original STS specification (Listing 1) and the revised STS specification (Listing 4) to design the relaxed resilience requirement. The STS designer wants to explore which threshold value of pollution can satisfy the property (Eq. (7)) with a probability of 0.85.

Figures 7, 8, 9, and 10 show the probabilities calculated for the original STS specification (Listing 1) and the revised STS specification (Listing 4) when the pollution threshold varying for a range of values and number of steps.

Figures 7, 8, 9, and 10 indicate that the original STS specification (Listing 1) fails to achieve minimum probability 0.85 for the property of Eq. (7) even after we increase the pollution threshold from 180 ppm to 240 ppm. For instance, a minimum probability of 0.73 is achieved when the pollution threshold is greater than 240 ppm within seven timesteps. However, Fig. 7 shows the minimum probability of 0.89 is achieved when the pollution threshold is greater than 180 ppm within seven timesteps for the revised STS specification (Listing 4). Similarly, minimum probabilities 0.93 (Fig. 8), 0.95 (Fig. 9), and 0.98 (Fig. 10) are achieved (within seven timesteps) when the pollution threshold is greater than 200 ppm, 220 ppm, and 240 ppm, respectively.

These threshold values of pollution provide an indication to the STS designer to express a new requirement. Figure 10 illustrates that 90% of the time, the system reduces the pollution below 100 ppm within five timesteps if the pollution threshold value is greater than 240 ppm (change in the operating environment). It means that the system will remain in the state where the pollution is below 100 ppm for the next seven, eight, nine, and ten timesteps with above 0.90 probability. Hence, the STS designer can design the new requirement for revised STS specification (Listing 4) as follows:

$$P_{min=?}[\mathsf{G} \le 7(pollution \ge 240 \Rightarrow P \ge 1[\mathsf{F}^{\le 5}(pollution \le 100)])] \tag{8}$$

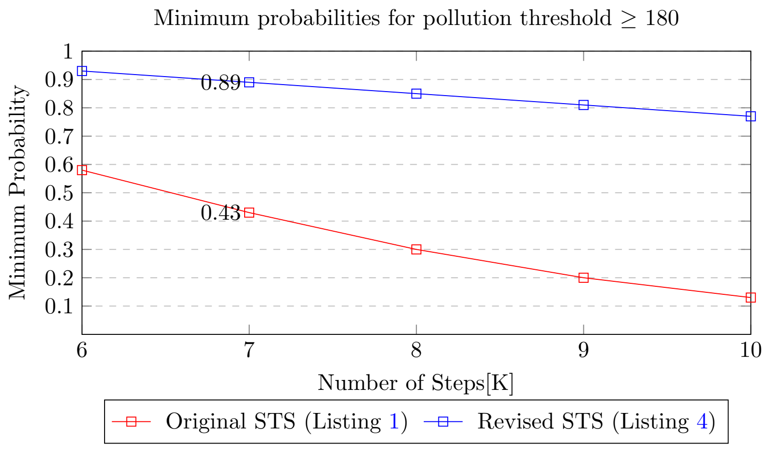

Fig. 7 Resilience (Eq. (7)) is not satisfied for the threshold value of pollution ≥ 180 when number of steps increased for original STS specification (Listing 1) but is satisfied for revised STS specification (Listing 4) within seven timesteps

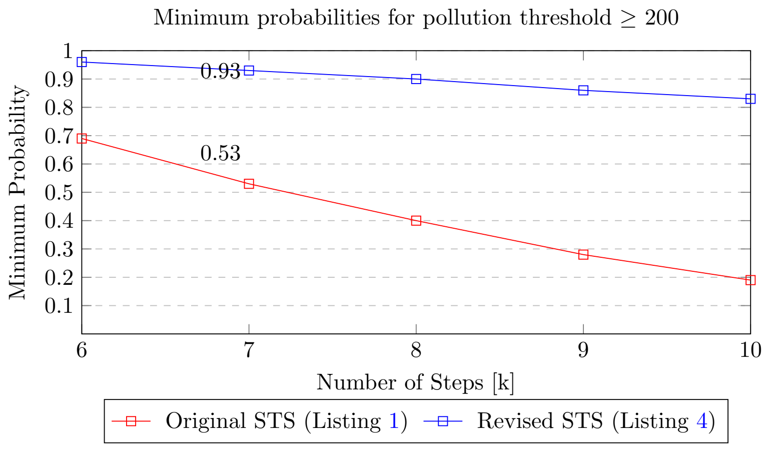

Fig. 8 Resilience (Eq. (7)) is not satisfied for the threshold value of pollution ≥ 200 when number of steps increased for original STS specification (Listing 1) but is satisfied for revised STS specification (Listing 4) within 7 timesteps

5.5 ReNo's Scalability

In this section, we will show computational cost of algorithms and scalability of ReNo. Section 5.5.1 has been used to show the time complexity of Algorithm 1. Section 5.5.2 has been used to show the time complexity of Algorithm 2. The scalability of ReNo has been explained in Sect. 5.5.3.

5.5.1 Analysis of time complexity for Algorithm 1

Theorem 2 *The time-complexity for the Algorithm 1 building PRISM model is* $\mathcal{O}(|A|^2 * |S| * |N|)$.

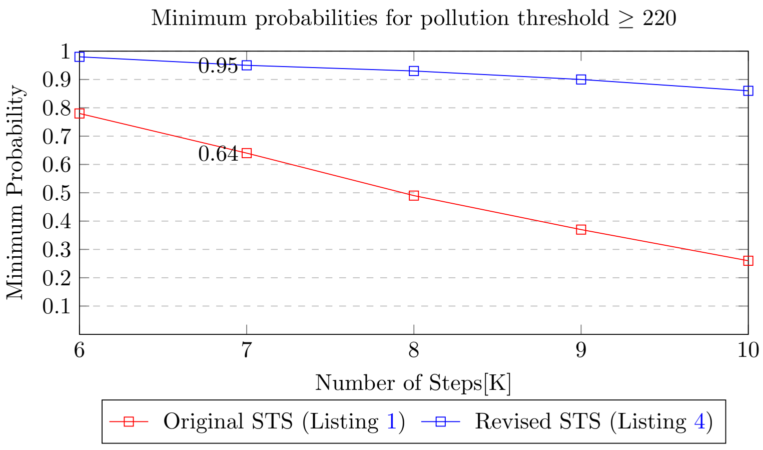

Fig. 9 Resilience (Eq. (7)) is not satisfied for the threshold pollution ≥ 220 when the number of steps increased for original STS specification (Listing 1) but is satisfied for revised STS specification (Listing 4) within 7 timesteps

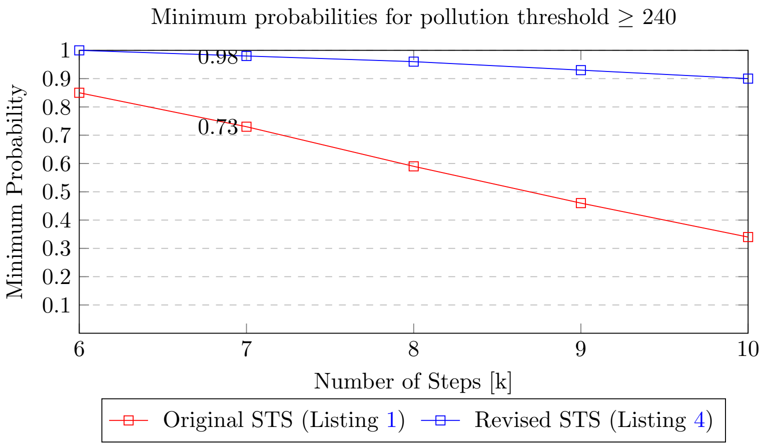

Fig. 10 Resilience (Eq. (7)) is not satisfied for the threshold value of pollution ≥ 180 when the number of steps increased for original STS specification (Listing 1) but is satisfied for revised STS specification (Listing 4) within seven timesteps

Proof From the initial state $s_0 \in S$, we consider each state in a set of states S. So, the outer loop executes at most $|S|$ times. Then, for each state, we consider each action in a set of actions A which means the inner loop executes at most $|A|$ * $|S|$ times. Then, for each action, we compute the state transition probability by multiplying the action selection probability and action execution probability. The time-complexity for Algorithm 2 computing selection probability in a given state s is $\mathcal{O}(|A| * |N|)$. Hence, Algorithm 2 executes at most $|A|$ * $|S|$ * $\mathcal{O}(|A| * |N|)$. Thus, the overall execution time is $\mathcal{O}(|A|^2 * |S| * |N|)$.

In a worst-case scenario, the number of states in the augmented probabilistic state transition model representing the PRISM model can increase exponentially with the number of state variables and their possible assignments. For example, if there are v state variables with n possible assignments each, the total number of states becomes $2^{(v*n)}$, which grows exponentially as v or n increases.

However, in our work, we have utilized the PRISM model checker programmatically to construct a PRISM model from an STS specification. PRISM employs Multi-Terminal Binary Decision Diagrams (MTBDDs) to represent the state space efficiently. MTBDDs offer a compact and symbolic representation of sets of states, transitions, and properties, which facilitates efficient state space exploration and model checking. Consequently, although the time complexity of MTBDD-based techniques can be exponential in theory, their practical performance is often significantly improved through optimization strategies like symbolic representation and efficient state space exploration. As a result, the complexities derived from theoretical worst-case analysis can be overly pessimistic for MTBDD-based techniques. To gain insights into the actual model statistics and performance, we conducted experiments, which are detailed in Section 5.5.3. □

5.5.2 Analysis of time complexity for Algorithm 2

Theorem 3 *The time-complexity for Algorithm 2 computing selection probability in a given state s is $\mathcal{O}(|A| * |N|)$.*

Proof In a given state s, we consider each action in a set of actions A. So, the outer loop executes at most $|A|$ times. Then for each action, we consider each norm in a set of norms N, which means the inner loop executes at most $|N|$ * $|A|$ times. Thus, the overall execution time is $\mathcal{O}(|A| * |N|)$. □

5.5.3 Model statistics

In this section, we analyse the scalability of our ReNo in terms of agents and their actions. All experiments were executed on a machine running Windows 10 Enterprise edition with an Intel® Core™ i7-7700 CPU with a 3.60GHz processor, 16.0 GB RAM, and a 64-bit Operating System. We performed all experiments on the STS specification in Listing 1, where each agent has three actions and three consequences except the null actions. We have used PRISM version 4.6.1 to conduct experiments but any PRISM version can be used to use our approach.

Table 4 shows the statistics of PRISM models we have built for an increasing number of agents and actions. Table 4 includes the number of agents and the number of actions executed by all agents. For example, there are four agents and twelve actions in Table 4 meaning that four agents all together execute twelve actions.

Table 4 also includes the number of states and transitions in the PRISM model. The construction time is the time to build the augmented probabilistic state transition model representing the PRISM model. The verification time is the time taken to verify the system property written in PCTL.

Overall, the construction time is calculated by combining the following: (1) the time taken by Algorithm 1 to generate a PRISM model from an STS specification, (2) the time taken by Algorithm 2 to compute the selection probabilities to select an action in the current state, (3) the time which is equivalent to the time taken for the system description to be parsed and converted to Multi-Terminal Binary Decision Diagrams (MTBDD) representing it, 4) the time to perform reachability, identify the non-reachable states and filter the MTBDD accordingly. We observe the state space explosion as the number of states of a model grows exponentially in terms of the number of variables and parallel components. Yet, they must be

Table 4 Model statistics

Agents & Actions		Model		Time (s)	
#Agents	#Actions	States	Transitions	Construction time (s)	Verification time (s)
4	12	100394	961958	264.8	9.419
8	24	143266	1316691	451.649	6.504
12	36	147633	1331552	463.472	17.105
16	48	308512	2931077	1377.17	18.057
20	60	319938	3050028	1442.601	21.732
24	72	340511	3252277	1545.675	31.903

represented in a limited computer memory in some form. For example, symbolic probabilistic model checking uses variants of binary decision diagrams (BDD) to compactly represent the state spaces of well-structured models in memory at the cost of verification runtime [45]. The complexity of PCTL model checking or requirement verification is linear in the size of a PCTL formula. (The size is defined as the number of logical connectives and temporal operators plus the sizes of temporal operators [45].)

However, as mentioned earlier, it is worth noting that the worst-case time complexity of building the PRISM model using Algorithm 1 can be exponential (see Sect. 5.5.1). Consequently, when dealing with large models, the computational time required for analysis may become excessively long. This can result in the impractical to complete the analysis within a reasonable timeframe.

6 Related work

We review relevant approaches to ReNo in the context of how to make a multiagent system more robust. Cheong and Winikoff [10] provide an approach to designing goal-oriented interactions that result in flexible and robust multiagent systems. In contrast, we define a sociotechnical system in computational terms as combining a social layer (characterized by norms) with a technical layer (characterized by actions and their effects). Our framework ReNo provides an approach to make a socio-technical system (STS) more resilient. Socio-technical systems are an important concept in the study of systems of autonomous systems, as studied in the fields of multiagent systems and decentralized AI. The significance of sociotechnical systems arises from how they synthesize social and technical layers to develop a more complete understanding of how AI agents may be developed and deployed to solve problems of societal significance.

Chopra et al. [12] consider the robustness of an organization with respect to business contracts. For them, a contract is robust if it helps lead to behaviours that satisfy an organization's goals and avoids undesirable outcomes. Their analysis is qualitative and concerns how an organization may be designed so it, for example, monitors and responds to potential or actual contract violations. In contrast, we are concerned with formal methods for evaluating a socio-technical system that includes multiple organizations.

More recent approaches [6, 14] have brought forth a notion of accountability in socio-technical systems. The idea is that, as stated above, the norms in an STS indicate what an agent may expect from another. Moreover, the agents can be held to accountable for any violations of the norms that may occur. An agent who violates a norm can be called upon to explain the violation or make corrections. Baldoni et al. [6] specify how responsibilities should be distributed in the STS. Chopra and Singh [14] show how capturing high-level requirements in terms of who is accountable for ensuring what can help create an STS that can better serve stakeholder needs by handling exception conditions more easily than systems where the requirements are at a low level. These approaches can benefit from the formal verification approach of this paper. If a multiagent system is verified with a certain probability to have the capability to recover before its execution, then this can be incorporated into the accountability mechanisms. Accountability mechanisms can be designed to monitor the performance of the system with respect to its requirements. For example, if the system fails to recover within the specified time frame, this can trigger an accountability mechanism that reports the failure to the relevant stakeholders and initiates a corrective action.

In the literature, multiagent systems have addressed exception handling, but they view it primarily as a mechanism for dealing with specific errors that may arise in the system. Hæg [24] proposes a fault-handling approach for multiagent systems that introduce special agents called sentinels. Sentinels act as a control structure over the system and provide a layer of fault tolerance. They do not participate in problem-solving but can intervene when necessary to choose alternative problem-solving methods. Similarly, Bungie [15] describes fault-tolerance patterns that can enhance the robustness of information protocols in a casual manner. Mandrake [16] is a programming model for decentralized applications that does not rely on infrastructure guarantees. It uses an information protocol that can be executed by agents in a shared-nothing environment using unreliable and unordered transport. Mandrake is focused on addressing faults that arise from violations of protocol expectations. In this context, fault handling can be seen as a form of exception handling, where agents with expectations can initiate recovery procedures to bring the system back into alignment with the protocol.

Additionally, Baldoni et al. [6] propose the notion of accountability as a means of achieving robustness in multiagent systems. The concept of accountability can aid in implementing effective exception-handling mechanisms. When an exceptional condition arises, the accountable agent(s) responsible for handling the situation is automatically notified with a corresponding account (i.e., exception) to take the necessary actions. Baldoni et al. [5] discuss how exception handling can be effectively integrated into distributed systems, which are implemented as multiagent systems, by utilizing the concept of responsibility at an organizational level. Gutierrez-Garcia's [23] approach in the context of normative multiagent systems involves modeling interaction protocols and exception handlers using obligations in deontic logic to handle exceptions.

Since norms are crucial to the functioning of an STS, it is important to consider where they come from. Aydoğan et al. [4] address how stakeholders can negotiate about which norms to adopt in their STS based on the stakeholders' values. Values here refer to the key beliefs and desires of an entity, such as privacy, freedom, and safety. The negotiation is facilitated by ontology-based reasoning to promote their respective value. Incorporating stakeholder values means that the stakeholders will try harder to satisfy the norms of the STS than if their values were ignored. In their agent-based negotiation framework [4], an agent revises the STS specification by revising a set of norm revision rules. However, their approach does not provide formal reasoning for resilience of the sort developed in our paper. Our framework, ReNo, offers a redesigned STS specification process that incorporates formal probabilistic reasoning. This enables us to satisfy stakeholders' requirements, including resilience, by adding or removing norms or by adding or removing agent actions. Our approach can potentially be enhanced to accommodate values by providing additional domain knowledge of how various actions performed by the agents promote or demote the values of the stakeholders of the STS and by including ontological reasoning about the application domain.

D'Inverno [20] is relevant to this work, insofar as they define a special class of multiagent systems called *electronic institutions* which resemble institutions in human societies. Their approach doesn't fully support autonomy in that a governor agent in an institution can overrule the decisions of a member agent. Additionally, their framework utilizes the state-based language Z to formally describe the behavior of a system in terms of its states and state transitions. State-based languages excel at capturing both the static structure and dynamic behavior of systems. However, to address the specific needs of probabilistic behaviors, our framework ReNo incorporates a temporal logic language PCTL. PCTL enables the specification and reasoning of properties in systems with probabilistic behaviors.

Ajmeri et al. [3] focus on how norms emerge in a decentralized manner based on the interactions of the agents and how they give (positive or negative) sanctions to each other based on their apparent norm violations. In their approach, an agent who violates a norm can share its context with other agents as a weak explanation of why it violated the norm. This context can help the receiving agent decide if the norm violation was justified in the specific context, affecting its sanctioning of the violator. Ajmeri et al. show that the norms that emerge lead to improved outcomes for the participating agents. Agrawal et al. [2] address similar intuitions but for explicit norms. Although Ajmeri et al. and Agrawal et al. provide useful intuitions about the emergence of norms, they do not produce norms of complex logical forms. Although their approach enables an individual agent to evaluate a specific execution run based on its local observation, they do not address the evaluation of an STS as a whole. Potentially, their approach could be used as another tool for an STS designer by using simulations to create richer norms. Our approach could be used to evaluate the norms (i.e., the STS) for resilience.

Gasparini et al. [22] have used the contrary-to-duty structure to design the normative specification and have used a preference relation over possible worlds that captures levels of system robustness. A qualitative reward function has been used to check the levels of compliance. In ReNo, MAS specification is defined as a social layer that includes norms and a technical layer that includes software components. Our framework uses probabilistic analysis to check resilience. Overall, Gasparini et al. address norm-driven multiagent planning in a probabilistic setting using Partially Observable Markov Decision Processes (POMDPs), and make provision for robustness analysis, but do not address the problems we pose.

Savarimuthu et al. [47–49] have provided an architecture to detect prohibition and obligation norms based on interactions between the agents. The prohibition and obligation detection algorithms utilizes association rule mining, a data mining technique, to identify sequences of events that may represent candidate norms. This architecture helps to modify or remove norms that are no longer relevant. Their work emphasises norm identification and has some relevance to our work. Our framework ReNo assumes the existence of norms and shifts the attention to verifying the resilience of a STS. In our approach, we evaluate whether an STS can effectively recover and meet resilience requirements.

Mahmoud et al. [35] propose a resource-aware adaptive punishment technique that enables norm establishment with larger neighbourhood sizes than resource-unaware punishment. Their evaluation of the adaptive technique has been done via a simulation. Dell'Anna et al. [19] propose a mechanism to revise the norms automatically that consider agents' preferences. The norms are revised by revising associated sanctions at runtime. The relationship between the satisfaction or violation of the norms and the achievement of the system-level objectives is learned from system execution data using a Bayesian Network. Then considering the runtime knowledge and the agent's preference, they develop heuristic strategies that automatically revise the sanctions associated with the enforced norms. They evaluate their mechanism on a traffic simulator ring-road environment. In ReNo, instead of focusing on sanctions, we examine commitment and prohibition norms. In terms of system properties, our focus is on probabilistic system properties, allowing for the consideration of uncertain system behavior. Within our framework, we have incorporated resilient requirements alongside achievement and maintenance properties to establish a resilient Socio-Technical System.

Kafalı et al. [27] propose a formal framework for the verification and refinement of the STS specification via social norms. They provide an agent-based simulation environment to mimic a social community consisting of multiple hospitals, physicians, and patients. They use an agent-based simulation to evaluate candidate STS designs and guide their refinement. A simulation environment is set up using parameters related to norms and mechanisms obtained from the STS specifications and requirements. Our approach is to design resilient sociotechnical systems by incorporating probabilistic model checking in a methodology for specifying a sociotechnical system and verifying time and quantity-constrained resilience requirements. We use probabilistic model checking to evaluate our approach.

Cámara and De Lemos [8] provide an approach for verifying self-adaptive systems. The focus is on resilience properties, which assess the system's ability to maintain reliable service provision despite environmental changes. They have considered Discrete-Time Markov Chains (DTMCs) as the probabilistic model employed to depict the system's behavior. The system's environment is stimulated for collecting execution traces to a build probabilistic model. PRISM is used to perform probabilistic model checking to evaluate the resilience of self-adaptive systems. Their framework has only relevance to our approach is verifying system property using probabilistic model checker PRISM. But our framework focuses on how to make resilient STS, by considering norms and actions. Model checking using PRISM combines probabilistic analysis and conventional reachability [32]. Unlike a simulation, our approach produces exact verification results of a property. Typically, there is a tradeoff: simulation gives us greater modeling realism, e.g., the ability to implement and experiment with various agent strategies, but it does not handle a formal notion of correctness. Model checking gives formal guarantees by checking properties against an STS model but requires a more abstract (simpler) model. Thus, model checking handles more limited phenomena, but with greater rigor.

Ostrom's notable contribution lies in her comprehensive focus on rules as the fundamental units of analysis in institutional theory and design. This emphasis highlights the vital role rules play in shaping institutional behavior and outcomes. Her work provides valuable insights into effective institutional design and management [43]. Social rules serve as the fundamental building blocks of institutional arrangements and are therefore essential elements in resilience analysis and design. They form the conceptual foundation upon which the analysis and design of resilient systems are built [42, 43]. Ostrom emphasizes that in the process of evolutionary institutional change, the variation in rules and norms is frequently a product of deliberate and rational design, rather than being driven solely by random factors. The work of Ostrom on the management of shared resources [44] is an example of how social regulation might provide a solution to problems such as the tragedy of the commons (the connection of such problems with normative multiagent systems has been made explicit in a number of papers, such as [46] and [7]). For instance, the authors [46] have used the perspective of self-governing institutions for common-pool resource (CPR) management, as defined by Ostrom, the concept of an institution encompasses the set of rules that outline the conditions governing the allocation and use of resources. These rules should be capable of being modified by additional rules, allowing for adaptation to the specific context in which the system operates. Additionally, the actions of individuals within the system can alter the environment, and external factors may also impact the system's dynamics [46]. Our framework RENO shares conceptual similarities to Ostrom's analysis and design of resilient systems, as outlined in her works ([42, 43]). The objective of our framework is to design and implement resilient socio-technical systems by adjusting either the social layer or the technical layer to fulfill the resilient requirements of stakeholders.

7 Conclusions and future directions

We propose ReNo, a probabilistic framework for the design and verification of STSs involving social norms, technical actions, and probabilistic temporal stakeholder requirements. Our main contribution is to introduce the idea of socio-technical resilience and incorporate probabilistic model checking in an overall methodology for specifying STSs and verifying resilience requirements. Normative behaviour has been incorporated into MDPs by considering norms as explicit constraints on agents' transitions [41], or by expressing norms explicitly as part of the state space and action space [21]. Other works on norm monitoring [18, 36] describe the expected normative behaviours of agents and resolved conflicts among norms using agents' objectives and preferences. We have incorporated norms explicitly into the PRISM model by calculating the probability of executing an action based on how the potential outcome of the action contributes towards the satisfaction or violation of the set of norms specified in the STS. We have also provided an algorithm that translates a formal STS specification into a probabilistic model. The ReNo framework provides a probabilistic guarantee that an STS meets its stakeholder requirements and recovers from failure within a certain period of time.

ReNo supports three core types of requirements: achievement and maintenance requirements are essential to the norm literature (e.g., a commitment to achieve something to a certain level or a prohibition to maintain something at a certain level). The third type is the resilience requirements (novel to ReNo) to verify that an STS can recover from requirement failures. We believe these three types of requirements cover realistic verification scenarios, to help guide STS designers. Our findings suggest that ReNo helps to build resilient STSs by demonstrating to the designer the tradeoffs between various STS specifications.

Future work

If a requirement is not satisfied, then counterexamples can reveal details about why the requirement failed and what modifications can be made to the STS specification to satisfy the requirement. PRISM can produce a violating adversary for a property, for which the verification outcome is false. We will explore employing such adversaries to guide the designer towards a revised STS.

Another important area of future development is the automation of the iterative process of STS re-design and requirements relaxation or modification based on the results of probabilistic model checking. Figuring out what relaxations are appropriate for stakeholders can involve sophisticated reasoning about goal conflicts [39] as well as creativity [37, 38] more generally. We currently offer methodological guidelines for doing this (to be followed by the STS designer and by stakeholders who specify requirements), but there is considerable reliance on human judgment for key decisions such as which STS re-designs or which requirements relaxations will lead us to a faster resolution of the problem of an STS design not realizing the specified requirements.

Acknowledgements We dedicate this article to the memory of Professor Aditya Ghose, who passed away unexpectedly in February 2023. This work was initiated with support from a grant from the University of

Wollongong and NC State University through a collaboration network called the University Global Partnership Network. MPS additionally thanks the US National Science Foundation (grant IIS-1908374) for partial support.

Appendix A java code: translation of an STS specification into a PRISM model

A.1 Code snippets for the initial state in the PRISM model

The Listing 5 shows java code to return the initial state of the PRISM model. The list variables have the agent variable and all state variables. The setValue(0, 1) in Line 2 in the Listing 5 shows that value 1 has been assigned to the variable at index 0 in variables list (here the variable agent is at index 0 in the list named variable). Similarly, random values have been assigned to other state variables.

Listing 5 Code Snippets for initial state in PRISM model construction

```
1 public State getInitialState() throws PrismException {
2 return new State(variables.size()).setValue(0, 3).setValue(1,20).setValue(2, 0); }
```

A.2 Code snippets to calculate the action selection probability

Listing 6 Code Snippets to calculate action selection probability

```
 for(int m=0;m<AgentsMechanism.get(a).size();m++) {
    if ((normType.equalsIgnoreCase("P")))  {
      if (n.getValue().containsKey(varM)) {
      int nConsquentValue = Integer.parseInt(n.getValue().get(AtomName));
      double currentTarget = nConsquentValue - valueOfVariable.get(varM) ;
      if (maxRange <= currentTarget) {
      prob = 1.0;
} else if (minRange >= currentTarget) {
      prob = 0.0;} else {
      prob = Double.parseDouble(new DecimalFormat("##.###").format((double) (currentTarget - minRange) / (maxRange - minRange)));}
      double result = Double.parseDouble(new DecimalFormat("##.###").format(Math.pow(w, prob)));
}
}
}
```

A.3 Code snippets for state transitions in the PRISM model

The Listing 7 shows the snippet of Java code of state transitions in the model. The function *computeTransitionTarget(int i, int offset)* is used to compute non-deterministic outcomes for executing all mechanisms of a selected agent. The Line 2 in the Listing 7 shows the current state. From Line 3 to Line 5 turns each agent using round-robin agent execution. In the initial state, the value of the agent variable is 1 (as explained in the use case) meaning that this agent is Company. In the next state, the value of the agent is 2 (i.e., Hospital); then the value of the agent is 3 (i.e., Regulator); and then the value of the agent is again 1 (i.e., Company). The remaining state variables are updated with new values in the Line 8 in the Listing 7.

The function *computeTransitionTarget(int i, int offset)* is used to compute non-deterministic outcomes of executing all mechanisms for a selected agent and the function *getTransitionProbability(int i, int offset)* is used to compute the state transition probability for each state transition.

Listing 7 Code Snippets for state transitions in PRISM model

```
1  public State computeTransitionTarget(int i, int
      offset) throws PrismException {
2  State target = new State(exploreState) ;
3   if((agent==numberofAgents)&&(sum!=0) )  {
      target.setValue(0,1);}
4  else if((sum!=0)){
5  target.setValue(0,agent+1);}
6  int newTotalValue =
      valueOfVariable.get(variables.get(p+1))
7  +Integer.parseInt(varValue[p]);
8  target.setValue(p+1, newTotalValue ) ;
9  return target;
10 }
11 public double getTransitionProbability(int i,
      int offset) throws PrismException {
12 int agent =
      valueOfVariable.get(variables.get(0));
13 for(int t=0;
      t<transitionProb.get(action).size();t++){
14 double probTra =
      transitionProb.get(action).get(t);
15 double prob = probTra * mechSelectionProb;
16                   return prob ;}
```

References

1. . Ciesinski, F. & Größer, M. (2004). in *On probabilistic computation tree logic* (eds Baier, C., Haverkort, B. R., Hermanns, H., Katoen, J.-P. & Siegle, M.) *Validation of Stochastic Systems* 147–188 (Springer, 2004). https://doi.org/10.1007/978-3-540-24611-4_5.
2. Agrawal, R., Ajmeri, N. & Singh, M. P. (2022). Socially intelligent genetic agents for the emergence of explicit norms. *Proceedings of the 31st International Joint Conference on Artificial Intelligence (IJCAI)* 10–16. https://doi.org/10.24963/ijcai.2022/2 .

3. Ajmeri, N., Guo, H., Murukannaiah, P. K. & Singh, M. P. (2018). Robust norm emergence by revealing and reasoning about context: Socially intelligent agents for enhancing privacy. *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)* 28–34. https://doi.org/10.24963/ijcai.2018/4 .
4. Aydoğan, R., Kafalı, Ö., Arslan, F., Jonker, C. M. & Singh, M. P. (2021). Nova: Value-based negotiation of norms. *ACM Transactions on Intelligent Systems and Technology (TIST)***12**(4), 45:1–45:29. https://doi.org/10.1145/3465054 .
5. Baldoni, M., Baroglio, C., Micalizio, R., & Tedeschi, S. (2022). Exception handling as a social concern. *IEEE Internet Computing, 26*(6), 33–40. https://doi.org/10.1109/MIC.2022.3216272
6. Baldoni, M., Baroglio, C., Micalizio, R., & Tedeschi, S. (2023). Accountability in multi-agent organizations: From conceptual design to agent programming. *Autonomous Agents and Multi-Agent Systems, 37*(1), 7. https://doi.org/10.1007/s10458-022-09590-6
7. Bevan, C. *et al.* (2013). Factors in the emergence and sustainability of self-regulation. *Social Coordination: Principles, Artefacts and Theories*. AISB Convention.
8. Cámara, J. & De Lemos, R. (2012). Evaluation of resilience in self-adaptive systems using probabilistic model-checking. *7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)* 53–62. https://doi.org/10.1109/SEAMS.2012.6224391 .
9. Castelfranchi, C. (1998). Modelling social action for AI agents. *Artificial Intelligence, 103*(1–2), 157–182. https://doi.org/10.1016/S0004-3702(98)00056-3
10. Cheong, C. & Winikoff, M. P. (2009). in *Hermes: Designing flexible and robust agent interactions* (ed.Dignum, V.) *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models* Ch. 5, 105–139 (IGI Global, Hershey, Pennsylvania, 2009). https://doi.org/10.4018/978-1-60566-256-5.ch005.
11. Chopra, A. K. & Singh, M. P. (2018). Sociotechnical systems and ethics in the large. *Proceedings of the AAAI/ACM Conference on Artificial Intelligence, Ethics, and Society (AIES)* 48–53. https://doi.org/10.1145/3278721.3278740 .
12. Chopra, A. K. *et al.* (2011). Analyzing contract robustness through a model of commitments. *Proceedings of the 11th International Workshop on Agent Oriented Software Engineering (AOSE 2010)* (6788), 17–36. https://doi.org/10.1007/978-3-642-22636-6_2 .
13. Chopra, A. K. *et al.* (2014). Protos: Foundations for engineering innovative sociotechnical systems. *Proceedings of the 22nd IEEE International Requirements Engineering Conference (RE)* 53–62. https://doi.org/10.1109/RE.2014.6912247 .
14. Chopra, A. K., & Singh, M. P. (2021). Accountability as a foundation for requirements in sociotechnical systems. *IEEE Internet Computing, 25*(6), 33–41. https://doi.org/10.1109/MIC.2021.3106835
15. Christie V, S. H., Chopra, A. K. & Singh, M. P. (2021). Bungie: Improving fault tolerance via extensible application-level protocols. *IEEE Computer***54**(5), 44–53. https://doi.org/10.1109/MC.2021.3052147 .
16. Christie, S. H., Chopra, A. K., & Singh, M. P. (2022). Mandrake: multiagent systems as a basis for programming fault-tolerant decentralized applications. *Autonomous Agents and Multi-Agent Systems, 36*(1), 16. https://doi.org/10.1007/s10458-021-09540-8
17. Dalpiaz, F., Giorgini, P., & Mylopoulos, J. (2013). Adaptive socio-technical systems: A requirements-based approach. *Requirements Engineering, 18*(1), 1–24. https://doi.org/10.1007/S00766-011-0132-1
18. Dastani, M., Torroni, P., & Yorke-Smith, N. (2018). Monitoring norms: A multi-disciplinary perspective. *Knowledge Engineering Review, 33*, e25. https://doi.org/10.1017/S0269888918000267
19. Dell'Anna, D., Dastani, M. & Dalpiaz, F. (2020). Runtime revision of sanctions in normative multi-agent systems. *Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS)***34** (2), 43.1–43.54. https://doi.org/10.1007/s10458-020-09465-8 .
20. d'Inverno, M., Luck, M., Noriega, P., Rodríguez-Aguilar, J. A., & Sierra, C. (2012). Communicating open systems. *Artificial Intelligence, 186*, 38–94. https://doi.org/10.1016/j.artint.2012.03.004
21. Fagundes, M. S., Ossowski, S., Luck, M., & Miles, S. (2012). Using normative Markov decision processes for evaluating electronic contracts. *AI Communications, 25*(1), 1–17. https://doi.org/10.3233/AIC-2012-0511
22. Gasparini, L., Norman, T. J., & Kollingbaum, M. J. (2018). Severity-sensitive norm-governed multi-agent planning. *Autonomous Agents and Multi-Agent Systems, 32*(1), 26–58. https://doi.org/10.1007/S10458-017-9372-X
23. Gutierrez-Garcia, J. O., Koning, J.-L. & Ramos-Corchado, F. F. (2009). An obligation approach for exception handling in interaction protocols. *2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology***3**, 497–500. https://doi.org/10.1109/WI-IAT.2009.334 .
24. Hägg, S. (1997). A sentinel approach to fault handling in multi-agent systems. *Multi-Agent Systems Methodologies and Applications: Second Australian Workshop on Distributed Artificial Intelli-*

gence Cairns, QLD, Australia, August 27, 1996 Selected Papers 2 181–195. https://doi.org/10.1007/BFB0030090 .

25. Hansson, H., & Jonsson, B. (1994). A logic for reasoning about time and reliability. *Formal Aspects of Computing, 6*(5), 512–535. https://doi.org/10.1007/BF01211866
26. Jones, A. J. I., Artikis, A., & Pitt, J. (2013). The design of intelligent socio-technical systems. *Artificial Intelligence Review, 39*(1), 5–20. https://doi.org/10.1007/s10462-012-9387-2
27. Kafalı, Ö., Ajmeri, N. & Singh, M. P. (2020). Specification of sociotechnical systems via patterns of regulation and control. *ACM Transactions on Software Engineering and Methodology (TOSEM)***29**(1), 7:1–7:50. https://doi.org/10.1145/3365664 .
28. Kafalı, Ö., Ajmeri, N., & Singh, M. P. (2016). Revani: Revising and verifying normative specifications for privacy. *IEEE Intelligent Systems (IS), 31*(5), 8–15. https://doi.org/10.1109/MIS.2016.89
29. Kampik, T. *et al.* (2022). Governance of autonomous agents on the web: Challenges and opportunities. *ACM Transactions on Internet Technology (TOIT)***22** (4), 104:1–104:31. https://doi.org/10.1145/3507910 .
30. Kwiatkowska, M., Norman, G. & Parker, D. (2002). PRISM: Probabilistic symbolic model checker. *Proceedings of the International Conference on Modelling Techniques and Tools for Computer Performance Evaluation* 200–204. https://doi.org/10.1007/3-540-46029-2_13 .
31. Kwiatkowska, M., Norman, G. & Parker, D. (2011). PRISM 4.0: Verification of probabilistic real-time systems. *Proceedings of the International Conference on Computer Aided Verification (CAV)* 585–591. https://doi.org/10.1007/978-3-642-22110-1_47 .
32. Kwiatkowska, M., Norman, G., & Parker, D. (2006). Quantitative analysis with the probabilistic model checker PRISM. *Electronic Notes in Theoretical Computer Science, 153*(2), 5–31.
33. Liaskos, S., Khan, S. M. & Mylopoulos, J. (2022). Modeling and reasoning about uncertainty in goal models: A decision-theoretic approach. *Software and Systems Modeling* 1–24. https://doi.org/10.1007/S10270-021-00968-W .
34. Mahala, G., Kafalı, O., Dam, H., Ghose, A. & Singh, M. P. (2023). Replication package. https://anonymous.4open.science/r/JAAMAS-ReNO-C6EE.
35. Mahmoud, S., Miles, S. & Luck, M. (2016). *Cooperation emergence under resource-constrained peer punishment*, 900–908 (International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), 2016). http://dl.acm.org/citation.cfm?id=2937056.
36. Modgil, S., et al. (2015). Monitoring compliance with e-contracts and norms. *Artificial Intelligence and Law, 23*(2), 161–196. https://doi.org/10.1007/s10506-015-9167-9
37. Murukannaiah, P. K., Ajmeri, N. & Singh, M. P. (2016). Acquiring creative requirements from the crowd: Understanding the influences of personality and creative potential in crowd RE. *Proceedings of the 24th IEEE International Requirements Engineering Conference (RE)* 176–185. https://doi.org/10.1109/RE.2016.68 .
38. Murukannaiah, P. K., Ajmeri, N. & Singh, M. P. (2022). Enhancing creativity as innovation via asynchronous crowdwork. *Proceedings of the 14th ACM Web Science Conference (WebSci)* 66–74. https://doi.org/10.1145/3501247.3531555 .
39. Murukannaiah, P. K., Kalia, A. K., Telang, P. R. & Singh, M. P. (2015). Resolving goal conflicts via argumentation-based analysis of competing hypotheses. *Proceedings of the 23rd IEEE International Requirements Engineering Conference (RE)* 156–165. https://doi.org/10.1109/RE.2015.7320418 .
40. Norman, T. J., & Reed, C. (2010). A logic of delegation. *Artificial Intelligence, 174*(1), 51–71. https://doi.org/10.1016/j.artint.2009.10.001
41. Oh, J., Meneguzzi, F., Sycara, K. & Norman, T. J. (2011). An agent architecture for prognostic reasoning assistance. *Proceedings of the 22nd International Joint Conference on Artificial Intelligence* 2513–2518. https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-418 .
42. Ostrom, E. (2008). in *Developing a method for analyzing institutional change* (eds Batie, S. & Mercuro, N.) *Alternative Institutional Structures* 66–94 (Routledge, 2008).
43. Ostrom, E. (2009). *Understanding Institutional Diversity* (Princeton University Press, 2009).
44. Ostrom, E. (2009). A general framework for analyzing sustainability of social-ecological systems. *Science, 325*(5939), 419–422. https://doi.org/10.1126/science.1172133
45. Parker, D. A. (2003). *Implementation of symbolic model checking for probabilistic systems*. Ph.D. thesis, University of Birmingham.
46. Pitt, J., Schaumeier, J., & Artikis, A. (2012). Axiomatization of socio-economic principles for self-organizing institutions: Concepts, experiments and challenges. *ACM Transactions on Autonomous and Adaptive Systems (TAAS), 7*(4), 1–39. https://doi.org/10.1145/2382570.2382575
47. Savarimuthu, B. T. R., Cranefield, S., Purvis, M. A. & Purvis, M. K. (2010). Obligation norm identification in agent societies. *Journal of Artificial Societies and Social Simulation***13** (4), 3. https://doi.org/10.18564/JASSS.1659 .

48. Savarimuthu, B. T. R., & Cranefield, S. (2011). Norm creation, spreading and emergence: A survey of simulation models of norms in multi-agent systems. *Multiagent and Grid Systems, 7*(1), 21–54. https://doi.org/10.3233/MGS-2011-0167
49. Savarimuthu, B. T. R., Cranefield, S., Purvis, M. A., & Purvis, M. K. (2013). Identifying prohibition norms in agent societies. *Artificial Intelligence and Law, 21*(1), 1–46. https://doi.org/10.1007/S10506-012-9126-7
50. Singh, A. M. & Singh, M. P. (2023). Norm deviation in multiagent systems: A foundation for responsible autonomy. *Proceedings of the 32nd International Joint Conference on Artificial Intelligence (IJCAI)* 289–297. https://doi.org/10.24963/ijcai.2023/33 .
51. Singh, M. P. (2013). Norms as a basis for governing sociotechnical systems. *ACM Transactions on Intelligent Systems and Technology (TIST)***5**(1), 21:1–21:23. https://doi.org/10.1145/2542182.2542203 .
52. Singh, M. P. (August 1990). in *Group ability and structure* (eds Demazeau, Y. & Müller, J.-P.) *Decentralized Artificial Intelligence, Volume 2* 127–145 (Elsevier/North-Holland, Amsterdam, 1991). Revised proceedings of the *2nd European Workshop on Modeling Autonomous Agents in a Multi Agent World (MAAMAW)*, St. Quentin en Yvelines, France.
53. Singh, M. P. (2014). Norms as a basis for governing sociotechnical systems. *ACM Transactions on Intelligent Systems and Technology (TIST), 5*(1), 1–23. https://doi.org/10.1145/2542182.2542203
54. Singh, A. M., & Singh, M. P. (2023). Wasabi: A conceptual model for trustworthy artificial intelligence. *IEEE Computer, 56*(2), 20–28. https://doi.org/10.1109/MC.2022.3212022
55. Verhagen, H., Neumann, M. & Singh, M. P. (2018). Normative multi-agent systems: Foundations and history. *Handbook of Normative Multiagent Systems. College Publications* 3–25. http://www.collegepublications.co.uk/downloads/handbooks00004.pdf.