Auton Agent Multi-Agent Syst (2009) 19:124–152
DOI 10.1007/s10458-008-9070-9

Normative conflict resolution in multi-agent systems

Wamberto W. Vasconcelos · Martin J. Kollingbaum · Timothy J. Norman

Published online: 9 November 2008

Abstract Norms (permissions, obligations and prohibitions) offer a useful and powerful abstraction with which to capture social constraints in multi-agent systems. Norms should exclude disruptive or antisocial behaviour without prescribing the design of individual agents or restricting their autonomy. An important challenge, however, in the design and management of systems governed by norms is that norms may, at times, conflict with one another; e.g, an action may be simultaneously prohibited and obliged for a particular agent. In such circumstances, agents no longer have the option of complying with these norms; whatever they do or refrain from doing will lead to a social constraint being broken. In this paper, we present mechanisms for the *detection* and *resolution* of normative conflicts. These mechanisms, based on first-order unification and constraint solving techniques, are the building blocks of more sophisticated algorithms we present for the management of normative positions, that is, the adoption and removal of permissions, obligations and prohibitions in societies of agents. We capture both direct and indirect conflicts between norms, formalise a practical concept of authority, and model conflicts that may arise as a result of delegation. We are able to formally define classic ways for resolving conflicts such as *lex superior* and *lex posterior*.

Keywords Norms · Detection and resolution of normative conflicts

W. W. Vasconcelos (✉) · T. J. Norman
Department of Computing Science, University of Aberdeen, Aberdeen AB24 3UE, UK
e-mail: wvasconcelos@acm.org

T. J. Norman
e-mail: t.j.norman@abdn.ac.uk

M. J. Kollingbaum
Robotics Institute, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA
e-mail: mkolling@cs.cmu.edu

1 Introduction

Techniques from multi-agent systems (MASs) offer the means to develop systems and integrate large, complex *systems of systems*[1] while maintaining the autonomy and heterogeneity of individual components [47]. One of the main challenges of MASs is to both *i)* maintain the autonomy and heterogeneity of system components (agents), and *ii)* provide guarantees (under appropriate assumptions) about the behaviour and outcomes of individual agents and of the system as a whole.

A promising approach to meeting this challenge advocates the explicit representation of norms—viz., the prohibitions, permissions and obligations—associated with individuals of a society [1,4,30]. Norms constrain the behaviour of norm-abiding agents—they will pursue their obligations, avoiding prohibitions, and taking advantage of their permissions. Our work described here deals with simple norms such as "agent x is obliged to pay" or "all auctioneer agents are permitted to start an auction". Conditional norms [5,25], that is, those such as "if agent x agrees to buy good y then it is obliged to pay for it", depicting specific circumstances and the resulting changes in normative positions associated with them, are not dealt with in this paper; however, we do sketch how our proposal can be extended to address them.

Norms offer a useful and powerful abstraction with which to specify and regulate multi-agent systems. They are designed to exclude disruptive or antisocial behaviour without prescribing the design of individual agents or restricting their autonomy [5]. Norms provide a generic account of individual behaviours; if all agents have norm-regulated behaviours, then we can offer guarantees about the multi-agent system. Similarly, norms provide a declarative statement of how agents are expected to behave, and so offer agents a way to predict the behaviour of others in response to requests, the provision of information, etc.

Normative specifications, however, raise a number of challenges. An important and little considered challenge is that normative positions may conflict. Normative conflicts may be simple; for example, an action may be simultaneously forbidden and obliged. Conflicts are not always easy to spot, however; for example, performing one action to meet an obligation may constitute the performance of another action (due to some domain theory) which is prohibited.[2]

Regardless of the complexity of the interactions between norms, the existence of a conflict will require resolution (or at least an informed choice of which norm to violate) for the agent to move forward; the agent will be in a position in which whatever it does (or refrains from doing) will violate a norm. We propose in this paper means to automatically detect normative conflicts and resolve them: our norms are manipulated in a finely-grained fashion, by the addition of constraints; the added constraint will prevent variables in norms' actions from having overlapping values.

Our approach to norm management contributes to the state-of-the-art in four important ways:

(1) We present a formal representation of norms with constraints.
(2) We present formal definitions of normative conflicts and define how they can be resolved, including indirect conflicts and those caused by agents delegating tasks.

[1] A system of systems is a "collection of task-oriented or dedicated systems that pool their resources and capabilities to obtain a new, more complex, 'system' which offers increased functionality and performance than simply the sum of the constituent systems", http://en.wikipedia.org/wiki/System_of_systems, accessed 5 May 2008 11:00am GMT.

[2] We are aware that inter-relationships among actions can be hard to establish in real-life scenarios. This effort would require techniques for knowledge capture/elicitation (e.g., [11]) leading to a domain theory in which dependencies among actions are formally forged.

(3) We present mechanisms for the adoption of new norms and the removal of norms to aid in the process of managing global normative states such that they remain conflict-free.
(4) We also present a formal account of authority, which is used to capture common phenomena such as *lex superior*.

Before presenting the perspective of norm-governed systems that we use in this paper (Sect. 3), we begin our exposition by informally introducing a scenario. The scenario outlines a rescue operation from a hazardous environment, and examples from this scenario are used throughout the paper to illustrate our formalisation and mechanisms for norm conflict detection and resolution. In Sect. 4, we formally define our notion of a normative conflict, explain how to resolve such a conflict, and present algorithms for conflict resolution. We then move on to investigating mechanisms for the management of global normative states by presenting algorithms for norm adoption and removal such that the normative state remains conflict-free (Sect. 5). In Sect. 6, we discuss indirect normative conflicts, that is, those arising due to relationships among disparate actions, and extend our approach to deal with such conflicts; conflicts caused by the delegation of norms among agents are also addressed. In Sect. 7, we provide a formal account of authority and how it relates to norms; we also formalise a notion of norm violation and sanction. We then revisit our scenario in some detail in Sect. 8 to complete our exposition and bring together the various threads of the paper in a coherent illustration of our contributions. We compare our approach to related work in Sect. 9 and we conclude the paper in Sect. 10, where we also reflect on our main contributions and give directions for future work.

2 Scenario: rescue operation

We illustrate our approach through the use of a simplified non-combatant evacuation scenario in which software agents help humans to coordinate their activities and information sharing. In this scenario there are two coalition partners, viz., team A and team B, operating within the same area, but each with independent assets. In our scenario, team A have received information that members of a non-governmental organisation (NGO) are stranded in a hazardous location. Intelligence has confirmed that these people must be evacuated to a safe location as soon as possible and that the successful completion of this operation takes highest priority.

Team A are based on an aircraft carrier just off the coast and have a number of assets at their disposal, including autonomous unmanned aerial vehicles (AUVs), deployed with sensors to provide on-going visual intelligence for the operation, and helicopters that can be deployed to rescue the NGO workers. Team B are located on land within close distance from the location of the NGO workers. The assets available to team B include ground troops and helicopters.

The most effective plan to complete the rescue mission is to deploy an AUV to provide real-time visual intelligence of the area in which the NGO workers are located, and then to dispatch the helicopter team to uplift the NGO workers and return them to the aircraft carrier. During the operation the AUV and the associated monitoring team on the carrier will provide the helicopter team with continuous updates regarding the location of the NGO workers and potential hazards in that area. Team A operate under the following norms:

– Team A commander is obliged to evacuate the stranded workers.
– Team A is forbidden to share AUV-obtained intelligence.
– Helicopters are forbidden to fly in bad weather.

After the AUV is dispatched to the target area, team A commander receives information from flight operations that the weather has deteriorated and it is no longer safe to deploy helicopters from the aircraft carrier. The alternative is to liaise with team B and deploy their troops and land-based helicopters, but to do so intelligence must be shared, and normative conflicts arise. In the ensuing sections we shall formalise this scenario and use it to illustrate the usefulness of our approach.

3 Norm-governed multi-agent systems

The design of complex MASs is greatly facilitated if we move away from individual components and, instead, regard them as belonging to stereotypical classes or categories of components. One way to carry out this classification/categorisation is through the use of *roles* as introduced in, e.g., [8,30]—an agent takes on a role within a society or an organisation, and this role defines a pattern of behaviour to which any agent ought to conform. For instance, within a humanitarian relief force, there are roles such as medical assistant, member of mine clearance team, and so on, and agents adopt these roles (possibly more than one) as they join the force. When agents adopt roles they commit themselves to the roles' expected behaviours, with associated sanctions and rewards.

We shall make use of two finite, non-empty sets, $Agents = \{a_1, \ldots, a_n\}$ and $Roles = \{r_1, \ldots, r_m\}$, representing, respectively, the sets of agent identifiers and role labels. We also need to refer to actions performed by agents:

Definition 1 An action tuple is $\langle a : r, \bar{\varphi}, t \rangle$ where

- $\bar{\varphi}$, a ground first-order atomic formula, representing an action
- $a \in Agents$ is the agent who did $\bar{\varphi}$
- $r \in Roles$ is the role played by the agent a when it did $\bar{\varphi}$
- $t \in \mathbb{N}$ is the time when $\bar{\varphi}$ was done

Agents perform their actions in a distributed fashion, contributing to the overall enactment of the MAS. However, for ease of presentation, we make use of a global (centralised) account for all actions taking place; therefore, it is important to record the authorship of actions and the time when they occur. We use the set Ξ to store the actions of agents—it represents a *trace* or a history of the enactment of a society of agents from a global point of view:

Definition 2 A global enactment state Ξ is a finite, possibly empty, set of action tuples $\langle a : r, \bar{\varphi}, t \rangle$.

A global enactment state Ξ can be "sliced" into many partial states $\Xi_a = \{\langle a : r, \bar{\varphi}, t \rangle \in \Xi \mid a \in Agents\}$ containing all actions of a specific agent a. Similarly, we could have partial states $\Xi_r = \{\langle a : r, \bar{\varphi}, t \rangle \in \Xi \mid r \in Roles\}$, representing the global state Ξ "sliced" across the various roles. We make use of a global enactment state to simplify our exposition; however, a fully distributed (and thus more scalable) account of enactment states can be achieved by slicing them as above and managing them in a distributed fashion.[3]

[3] In [14] we present a distributed architecture for electronic institutions [8], in which global enactment states are broken down into *scenes*, that is, agent sub-activities with specific purposes, such as the registration process in a virtual auction room, the auction itself and the settlement of bills (and delivery of goods).

3.1 A representation for norms

In this section we introduce our representation of norms. We extend our previous work [41], adopting the notation of [30] for specifying norms, complementing it with *constraints* [18]. Constraints are used to further *refine* the scope of influence of norms on actions, and are thus defined:

Definition 3 Constraints, represented as γ, are any construct of the form $\tau \lhd \tau'$, where τ, τ' are first-order terms (that is, a variable, a constant or a function applied to terms) and $\lhd \in \{=, \neq, >, \geq, <, \leq\}$.

We shall denote a possibly empty set of constraints as $\Gamma = \{\gamma_0, \ldots, \gamma_n\}$ and it stands as a *conjunction* of the constraints, that is, $\bigwedge_{i=0}^{n} \gamma_i$. We make use of numbers and arithmetic functions to build terms τ. Arithmetic functions may appear infix, following their usual conventions. We adopt Prolog's convention [2] and use strings starting with a capital letter to represent variables and strings starting with a small letter to represent constants. Some sample constraints are $X < 120$ and $X < (Y + Z)$. To improve readability, constraints of the form $\{10 \leq X, X \leq 45\}$ will be written as $\{10 \leq X \leq 45\}$.

In our work, constraints are associated with first-order formulae, imposing restrictions on their variables. We represent this association as $\varphi \circ \Gamma$, as in, for instance, $deploy(s_1, X, Y) \circ \{10 \leq X \leq 50, 5 \leq Y \leq 45\}$. When Γ is empty, we will simply drop it from our formulae. Norms are thus defined:

Definition 4 A norm ω is a tuple $\langle \nu, t_d, t_a, t_e \rangle$, with ν being either

- $\mathsf{O}_{\alpha:\rho}\varphi \circ \Gamma$ (an obligation),
- $\mathsf{P}_{\alpha:\rho}\varphi \circ \Gamma$ (a permission), or
- $\mathsf{F}_{\alpha:\rho}\varphi \circ \Gamma$ (a prohibition),

where α, ρ are terms, $\varphi \circ \Gamma$ is a first-order atomic formula with constraints; $t_d, t_a, t_e \in \mathbb{N}$ are, respectively, the time when ν was introduced, when ν becomes active and when ν expires, $t_d \leq t_a \leq t_e$.

Term α identifies the agent(s) to whom the norm is applicable and ρ is the role of such agent(s). $\mathsf{O}_{\alpha:\rho}\varphi \circ \{\gamma_0, \ldots, \gamma_n\}$ thus represents an obligation on agent α taking up role ρ to bring about φ, subject to *all* constraints γ_i, $0 \leq i \leq n$. The γ_i terms express constraints on variables of φ.

For simplicity, in our discussion we assume an implicit universal quantification over variables in ν. For instance, $\mathsf{P}_{A:R}deploy(X, b, c)$ stands for $\forall A \in Agents.\forall R \in Roles.\forall X.$ $\mathsf{P}_{A:R}deploy(X, b, c)$. However, our proposal can be naturally extended to cope with arbitrary quantifications. Obligations normally require the arguments of their actions to be existentially quantified, as in, for instance

$$\forall A \in Agents.\forall R \in Roles.\exists X.\exists Y.\exists Z.\mathsf{O}_{A:R}deploy(X, Y, Z)$$

Quantifications on agent ids and role labels may be universal or existential, and the relative ordering of quantifications defines the applicability of the norm, following the usual first-order logic semantics [10,26].

We propose to formally represent from a global perspective the normative positions [35] of all agents taking part in a virtual society. By "normative position" we mean the "social burden" associated with individuals [14], that is, their obligations, permissions and prohibitions:

Definition 5 A global normative state is a finite and possibly empty set $\Omega = \{\omega_0, \ldots, \omega_n\}$, of norms ω_i, $0 \leq i \leq n$.

A global normative state Ω complements the enactment state Ξ of a virtual society, with information on the normative positions of individual agents. The management (i.e., creation and updating) of global normative states is an interesting area of research. A practical approach is that of [15]: rules depict how norms should be inserted and removed as a result of agents' actions. A sample rule is

$$\langle Ag_1 : cmdr, ask(Ag_2, Info), T\rangle \rightsquigarrow \oplus\langle \mathsf{O}_{Ag_2:int} supply(Ag_1, Info), (T+1), (T+1), (T+5)\rangle$$

representing that if an agent Ag_1 acting as a commander asks Ag_2 some *Info* at time T then we introduce (denoted by the "$\oplus$" operator) an obligation at time $T+1$ (and with immediate effect) on Ag_2 acting as an intelligence officer, to supply Ag_1 the requested *Info* within 5 "ticks" of the clock. Similarly to Ξ, we use a single normative state Ω to simplify our exposition; we can also slice Ω into various sub-sets and manage them in a distributed fashion as explored in [13].

3.2 Substitutions, unification and constraint satisfaction

We use first-order unification [10] and constraint satisfaction [18] as the building blocks of our mechanisms. Unification allows us (i) to detect whether norms are in conflict and (ii) to detect the set of actions that are under the influence of a norm. Initially, we define substitutions:

Definition 6 A substitution σ is a finite and possibly empty set of pairs x/τ, where x is a variable and τ is a term.

We define the application of a substitution in accordance with [10]. In addition, we describe how substitutions are applied to sets of constraints and norms (X stands for O, P or F):

(1) $c \cdot \sigma = c$ for a constant c.
(2) $x \cdot \sigma = \tau \cdot \sigma$ if $x/\tau \in \sigma$; otherwise $x \cdot \sigma = x$.
(3) $p^n(\tau_0, \ldots, \tau_n) \cdot \sigma = p^n(\tau_0 \cdot \sigma, \ldots, \tau_n \cdot \sigma)$.
(4) $\{\gamma_0, \ldots, \gamma_n\} \cdot \sigma = \{\gamma_0 \cdot \sigma, \ldots, \gamma_n \cdot \sigma\}$
(5) $(\mathsf{X}_{\alpha:\rho}\varphi \circ \Gamma) \cdot \sigma = \big(\mathsf{X}_{(\alpha\cdot\sigma):(\rho\cdot\sigma)}(\varphi \cdot \sigma) \circ (\Gamma \cdot \sigma)\big)$.
(6) $\langle v, t_d, t_a, t_e\rangle \cdot \sigma = \langle (v \cdot \sigma), t_d, t_a, t_e\rangle$

A substitution σ is a *unifier* of two terms τ_1, τ_2, if $\tau_1 \cdot \sigma = \tau_2 \cdot \sigma$. Unification is a fundamental problem in automated theorem proving and many algorithms have been proposed [10]; recent work offers means to obtain unifiers efficiently. We use unification in the following way:

Definition 7 $unify(\tau_1, \tau_2, \sigma)$ holds iff $\tau_1 \cdot \sigma = \tau_2 \cdot \sigma$, for some σ. $unify(p^n(\tau_0, \ldots, \tau_n), p^n(\tau'_0, \ldots, \tau'_n), \sigma)$ holds iff $unify(\tau_i, \tau'_i, \sigma), 0 \leq i \leq n$.

The *unify* relationship checks if a substitution σ is indeed a unifier for τ_1, τ_2, but it can also be used to find σ. We assume that *unify* is a suitable implementation of a unification algorithm which (i) always terminates (possibly failing, if a unifier cannot be found); (ii) is correct; and (iii) has a linear computational complexity.

We make use of existing constraint satisfaction techniques [17,18] to implement a *satisfy* predicate which checks if a given set of constraints admits one solution, that is, the predicate holds if the variables of the constraints admit at least one value which simultaneously fulfils all constraints:

Definition 8 $satisfy(\{\gamma_0, \ldots, \gamma_n\})$ holds iff $\bigwedge_{i=0}^{n}(\gamma_i \cdot \sigma)$ is true for some σ.

$$
\begin{array}{ll}
1 & \mathit{inScope}(\mathit{Action}, \omega) \leftarrow \\
2 & \quad \mathit{Action} = \langle a : r, \bar{\varphi}, t \rangle \wedge \\
3 & \quad \omega = \langle \mathsf{X}_{\alpha:\rho} \varphi \circ \Gamma, t_d, t_a, t_e \rangle \wedge \\
4 & \quad \mathit{unify}(\langle a, r, \bar{\varphi} \rangle, \langle \alpha, \rho, \varphi \rangle, \sigma) \wedge \\
5 & \quad \mathit{satisfy}(\Gamma \cdot \sigma) \wedge \\
6 & \quad t_a \leq t \leq t_e
\end{array}
$$

Fig. 1 Check if action is within influence of a norm

This predicate can be implemented via different "off-the-shelf" constraint satisfaction libraries; for instance, it can be defined via the built-in `call_ residue_vars/2` predicate, available in SICStus Prolog [38] as:

$$\mathit{satisfy}(\{\gamma_0, \ldots, \gamma_n\}) \leftarrow \texttt{call\_residue\_vars}((\gamma_0, \ldots, \gamma_n), \_)$$

Predicate `call_residue_vars`(*Goals*, *Vars*) evaluates if *Goals* admit one possible solution, collecting in *Vars* the list of residual variables that have blocked goals or attributes attached to them. In our definition above, the value of *Vars* is not relevant, as we simply want to know if *Goals* are satisfiable.

3.3 Meaning of norms

We explain the meaning of our norms in terms of their relationships with action tuples of global enactment states. We define when an individual action tuple is within the scope of influence of a norm—we do so via the logic program of Fig. 1. It defines predicate *inScope* which holds if its first argument, an action tuple (in the format of the elements of Ξ of Def. 2), is within the influence of a norm ω (in the format of Def. 4), its second parameter. Lines 2 and 3 define, respectively, the format of *Action* and ω (where X is either P, F or O). Line 4 tests (i) if the agent performing the action and its role unify with α, ρ of ω and (ii) if the actions $\bar{\varphi}$ and φ unify. Line 5 checks if the constraints on ω (instantiated with the substitution σ obtained in line 4) can be satisfied, and, finally, line 6 checks if the time of the action is within the norm's time frame.

A global enactment state Ξ at instant t_{now} is compliant with a global normative state Ω if both conditions below hold:

- All active prohibitions in Ω (i.e., those that have not yet expired) do not have an action within their scope, that is,

$$\forall\, \omega \in \Omega . \forall \mathit{Action} \in \Xi . \omega = \langle \mathsf{F}_{\alpha:\rho} \varphi \circ \Gamma, t_d, t_a, t_e \rangle \wedge t_e \geq t_{now} \rightarrow \neg \mathit{inScope}(\mathit{Action}, \omega)$$

- All expired obligations have a corresponding action within their scope, that is,

$$\forall\, \omega \in \Omega . \exists \mathit{Action} \in \Xi . (\omega = \langle \mathsf{O}_{\alpha:\rho} \varphi \circ \Gamma, t_d, t_a, t_e \rangle \wedge t_e < t_{now}) \rightarrow \mathit{inScope}(\mathit{Action}, \omega)$$

Permissions can be interpreted differently depending on how much autonomy we want agents to have [16]. For instance, one might insist that only explicitly permitted actions are performed—more formally:

$$\forall \mathit{Action} \in \Xi . \exists \omega \in \Omega . \mathit{Action} = \langle a : r, \bar{\varphi}, t \rangle \rightarrow \begin{pmatrix} \omega = \langle \mathsf{P}_{\alpha:\rho} \varphi \circ \Gamma, t_d, t_a, t_e \rangle \wedge \\ \mathit{inScope}(\mathit{Action}, \omega) \end{pmatrix}$$

Agents may experience difficulties if an action is simultaneously within the scope of influence of a prohibition and an obligation (or a prohibition and a permission). In such circumstances, whatever the agents do or refrain from doing, may give rise to an enactment state that is not

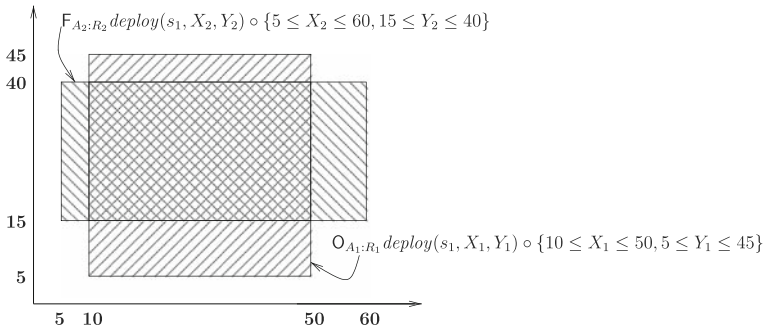

Fig. 2 Conflict detection: overlap in scopes of influence

norm-compliant. The agents will thus violate a norm, and will be subject to sanctions—in Sect. 7.3 we explain how norm violations/sanctions are dealt with.

If an agent has a set of candidate actions subject to a set of conflict-free norms, then predicate *inScope* can be used to select among the actions, namely those that are not within the scope of any prohibitions. Alternatively, agents can use the mechanism above to select those actions that are within the scope of obligations, and hence should be given priority. These strategies have been explored in [14].

4 Norm conflicts

This section provides definitions for norm conflicts, enabling their detection and resolution. Constraints confer more expressiveness and precision on norms, but mechanisms for detection and resolution must factor them in.

4.1 Conflict detection

A conflict arises when an action is simultaneously prohibited and permitted/obliged, and its variables have overlapping values. The variables of a norm specify its scope of influence, that is, which agent/role the norm concerns, and which values of the action it addresses. In Fig. 2, we show two norms over action $deploy(S, X, Y)$, establishing that sensor S is to be deployed on grid position (X, Y). The norms are

$$\mathsf{O}_{A_1:R_1} deploy(s_1, X_1, Y_1) \circ \{10 \leq X_1 \leq 50, 5 \leq Y_1 \leq 45\}$$
$$\mathsf{F}_{A_2:R_2} deploy(s_1, X_2, Y_2) \circ \{5 \leq X_2 \leq 60, 15 \leq Y_2 \leq 40\}$$

Their scopes are shown as rectangles filled with different patterns. The overlap of their scopes is the rectangle in which both patterns are superimposed. Norm conflict is formally defined as follows:

Definition 9 Norms $\omega, \omega' \in \Omega$, are in conflict under substitution σ, denoted as $conflict(\omega, \omega', \sigma)$, X being O or P, iff:

- $\omega = \langle \mathsf{F}_{\alpha:\rho} \varphi \circ \Gamma, t_d, t_a, t_e \rangle, \omega' = \langle \mathsf{X}_{\alpha':\rho'} \varphi' \circ \Gamma', t'_d, t'_a, t'_e \rangle$ or
- $\omega = \langle \mathsf{X}_{\alpha:\rho} \varphi \circ \Gamma, t_d, t_a, t_e \rangle, \omega' = \langle \mathsf{F}_{\alpha':\rho'} \varphi' \circ \Gamma', t'_d, t'_a, t'_e \rangle$

and the following conditions hold:

(1) $unify(\langle \alpha, \rho, \varphi \rangle, \langle \alpha', \rho', \varphi' \rangle, \sigma)$,
(2) $satisfy((\Gamma \cup \Gamma') \cdot \sigma)$ and
(3) $overlap(t_a, t_e, t'_a, t'_e)$

That is, a conflict occurs between a prohibition and either an obligation or a permission if (1) a substitution σ can be found that unifies the variables of the two norms, (2) the constraints from both norms can be satisfied (taking σ under consideration), and (3) the activation period of the norms overlap—the *overlap* relationship holds if $t_a \leq t'_a \leq t_e$ or $t'_a \leq t_a \leq t'_e$.

The norm conflict of Fig. 2 is indeed captured by Definition 9. We can obtain a substitution $\sigma = \{X_1/X_2, Y_1/Y_2\}$ and this is a first indication that there may be a conflict or *overlap* of influence between both norms regarding the defined action. The constraints on the norms may restrict the overlap and, therefore, leave actions under certain variable bindings free of conflict. We, therefore, have to investigate the constraints of both norms in order to see if an overlap of the values indeed occurs. In our example, the obligation has constraints $\{10 \leq X_1 \leq 50, 5 \leq Y_1 \leq 45\}$ and the prohibition has constraints $\{5 \leq X_2 \leq 60, 15 \leq Y_2 \leq 40\}$. By using the substitutions we can "merge" the constraints as $\{10 \leq X_2 \leq 50, 5 \leq X_2 \leq 60, 5 \leq Y_2 \leq 45, 15 \leq Y_2 \leq 40\}$; the overlap of the merged constraints is $10 \leq X_2 \leq 60$ and $15 \leq Y_2 \leq 40$ and they represent ranges of values for variables X_1, X_2 and Y_1, Y_2 where a conflict will occur.

For convenience (and without any loss of generality), we assume that our norms are in a special format: all terms τ occurring in ν are replaced by a fresh variable x (not occurring anywhere in ν) and a constraint $x = \tau$ is added to Γ. This is an extended form of *explicit unification* [36] and the transformation of formulae from their usual format to this extended explicit unification format can be easily automated by scanning ν from left to right, collecting all terms $\{\tau_1, \ldots, \tau_n\}$; then we add $\{x_1 = \tau_1, \ldots, x_n = \tau_n\}$ to Γ. For example, norm $\mathsf{P}_{A:R} deploy(s_1, X, Y) \circ \{X > 50\}$ becomes $\mathsf{P}_{A':R'} deploy(S, X', Y') \circ \{A' = A, R' = R, S = s_1, X' = X, Y' = Y, X > 50\}$. Although some of the added constraints $x = y$ may seem superfluous, they are required to ensure that unconstrained variables are properly dealt by our conflict resolution mechanism presented below.

4.2 Conflict resolution

We resolve conflicts by manipulating the constraints associated to the norms' variables, removing any overlap in their values. In Fig. 3 we show the norms of Fig. 2 without the intersection between their scopes of influence[4]—the prohibition has been *curtailed*, its scope being reduced to avoid the values that the obligation addresses. Specific constraints are added to the prohibition in order to perform this curtailment; these additional constraints are derived from the obligation, as we explain below. In our example, we obtain two prohibitions, viz., $\mathsf{F}_{A_2:R_2} deploy(s_1, X_2, Y_2) \circ \{5 \leq X_2 < 10, 15 \leq Y_2 \leq 40\}$ and $\mathsf{F}_{A_2:R_2} deploy(s_1, X_2, Y_2) \circ \{50 < X_2 \leq 60, 15 \leq Y_2 \leq 40\}$.

We formally define below how the curtailment of norms takes place. It is important to notice that the curtailment of a norm creates a new set Ω of curtailed norms:

Definition 10 Relationship $curtail(\omega, \omega', \Omega)$, where

- $\omega = \langle \mathsf{X}_{\alpha:\rho} \varphi \circ \{\gamma_0, \ldots, \gamma_n\}, t_d, t_a, t_e \rangle$ and
- $\omega' = \langle \mathsf{X}'_{\alpha':\rho'} \varphi' \circ \{\gamma'_0, \ldots, \gamma'_m\}, t'_d, t'_a, t'_e \rangle$

X and X' being either O, F or P, holds iff Ω is a possibly empty and finite set of norms obtained by curtailing ω with respect to ω'. The following cases arise:

[4] For clarity, in this example we show the norms in their usual format without explicit unifications.

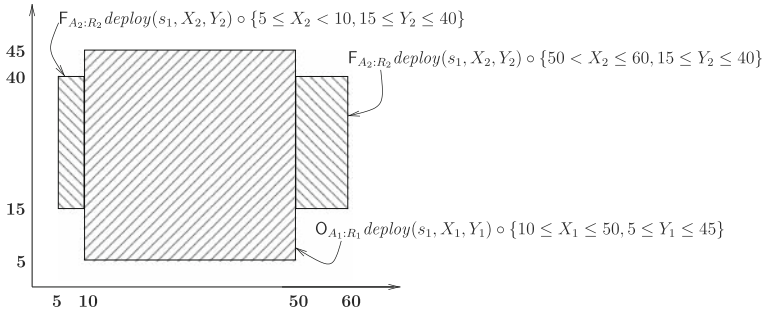

Fig. 3 Conflict resolution: curtailment of scopes of influence

(1) If $conflict(\omega, \omega', \sigma)$ does not hold then $\Omega = \{\omega\}$; that is, the curtailment of a non-conflicting norm ω is ω itself.

(2) If $conflict(\omega, \omega', \sigma)$ holds, then $\Omega = \{\omega_0^c, \ldots, \omega_m^c\}$, where $\omega_j^c = \langle \mathsf{X}_{\alpha:\rho}\varphi \circ \big(\{\gamma_0, \ldots, \gamma_n\} \cup \{\neg(\gamma_j' \cdot \sigma)\}\big), t_d, t_a, t_e\rangle$, $0 \leq j \leq m$.

In order to curtail ω, thus avoiding any overlapping of the values its variables may have with those variables of ω', we must "merge" the negated constraints of ω' with those of ω. Additionally, in order to ensure the appropriate correspondence of variables between ω and ω' is captured, we must apply the substitution σ obtained via $conflict(\omega, \omega', \sigma)$ on the merged negated constraints.

We combine the constraints of $\nu = \mathsf{X}_{\alpha:\rho}\varphi \circ \{\gamma_0, \ldots, \gamma_n\}$ with the negated constraints of $\nu' = \mathsf{X}'_{\alpha':\rho'}\varphi' \circ \{\gamma_0', \ldots, \gamma_m'\}$. If we regard the set of constraints as a conjunction of constraints, that is, $\{\gamma_0, \ldots, \gamma_i\}$ is seen as $\bigwedge_{i=0}^{n} \gamma_i$, and if we regard "$\circ$" as the conjunction operator $\wedge$, then the following equivalences hold

$$\mathsf{X}_{\alpha:\rho}\varphi \wedge \left(\bigwedge_{i=0}^{n} \gamma_i \wedge \neg\left(\bigwedge_{j=0}^{m} \gamma_j' \cdot \sigma\right)\right) \equiv \mathsf{X}_{\alpha:\rho}\varphi \wedge \left(\bigwedge_{i=0}^{n} \gamma_i \wedge \left(\bigvee_{j=0}^{m} (\neg\gamma_j' \cdot \sigma)\right)\right)$$

We can rewrite the last formula as

$$\bigvee_{j=0}^{m}\left(\mathsf{X}_{\alpha:\rho}\varphi \wedge \left(\bigwedge_{i=0}^{n} \gamma_i \wedge \neg(\gamma_j' \cdot \sigma)\right)\right)$$

That is, each constraint on ν' leads to a possible solution for the resolution of a conflict and a possible curtailment of ν, as it prevents the overlap among variables. The curtailment thus produces a set of curtailed norms

$$\bigcup_{j=0}^{m} \nu_j^c = \bigcup_{j=0}^{m} \{\mathsf{X}_{\alpha:\rho}\varphi \circ (\{\gamma_0, \ldots, \gamma_n\} \cup \{\neg(\gamma_j' \cdot \sigma)\})\}$$

Although each of the ν_j^c, $0 \leq j \leq m$, represents a solution to the norm conflict, *all* of them are added to Ω in order to replace the curtailed norm. This allows the preservation of as much of the original scope of the curtailed norm as possible. Figure 3 illustrates this: the result of the curtailment are two new prohibitions applicable to all those coordinates of the original prohibition which are not covered by the obligation, rather than just one of them. However, replacing the original prohibition with one of its curtailed versions would resolve the conflict.

Let us consider a further example. Suppose that we have two norms, ω and ω', that are in conflict. Norm ω states that it is forbidden for any agent in any role to deploy sensor s_1 in any location such that X is 100 and Y is greater than 50; this prohibition was issued at time t_d and is in effect between t_a and t_e. Norm ω' also refers to the deployment of sensor s_1, but permits agent ag_1 playing role r_1 to deploy this sensor in certain locations that overlap with those prohibited by ω; ω' is in effect during the same period of time as ω. These norms are represented as follows:

$$\omega = \left\langle \mathsf{F}_{A:R} deploy(S, X, Y) \circ \left\{ \begin{array}{c} S = s_1, \\ X = 100, \\ Y > 50 \end{array} \right\}, t_d, t_a, t_e \right\rangle$$

$$\omega' = \left\langle \mathsf{P}_{A':R'} deploy(S', X', Y') \circ \left\{ \begin{array}{c} A' = ag_1, \\ R' = r_1, \\ S' = s_1, \\ X' = 100, \\ Y' > 100 \end{array} \right\}, t'_d, t_a, t_e \right\rangle$$

These norms are in conflict since $conflict(\omega, \omega', \{A'/A, R'/R, S'/S, X'/X, Y'/Y\})$ holds. We can resolve this conflict by curtailing ω with constraints $A' = ag_1$, $R' = r_1$, $S' = s_1$ $X' = 100$ and $Y' > 100$ of ω'. Following Definition 10, we obtain the set Ω of curtailed norms:

$$\left\{ \begin{array}{l} \langle \mathsf{F}_{A:R}\varphi \circ (\Gamma \cup \{\neg(A = ag_1)\}), t_d, t_a, t_e\rangle, \\ \langle \mathsf{F}_{A:R}\varphi \circ (\Gamma \cup \{\neg(R = r_1)\}), t_d, t_a, t_e\rangle, \\ \langle \mathsf{F}_{A:R}\varphi \circ (\Gamma \cup \{\neg(S = s_1)\}), t_d, t_a, t_e\rangle, \\ \langle \mathsf{F}_{A:R}\varphi \circ (\Gamma \cup \{\neg(X = 100)\}), t_d, t_a, t_e\rangle, \\ \langle \mathsf{F}_{A:R}\varphi \circ (\Gamma \cup \{\neg(Y > 100)\}), t_d, t_a, t_e\rangle \end{array} \right\}$$

We show the original prohibition with the added negated constraints stemming from ω'—each negated constraint giving rise to a curtailed prohibition; the substitution σ ensures that the constraints coming from ω' are indeed associated with the corresponding variables of ω (which are shown above already unified with their respective values). We can present the set above in a clearer alternative format:

$$\left\{ \begin{array}{l} \langle \mathsf{F}_{A:R}\varphi \circ (\Gamma \cup \{A \neq ag_1\}), t_d, t_a, t_e\rangle, \\ \langle \mathsf{F}_{A:R}\varphi \circ (\Gamma \cup \{R \neq r_1\}), t_d, t_a, t_e\rangle, \\ \langle \mathsf{F}_{A:R}\varphi \circ (\Gamma \cup \{S \neq s_1\}), t_d, t_a, t_e\rangle, \\ \langle \mathsf{F}_{A:R}\varphi \circ (\Gamma \cup \{X \neq 100\}), t_d, t_a, t_e\rangle, \\ \langle \mathsf{F}_{A:R}\varphi \circ (\Gamma \cup \{Y \leq 100\}), t_d, t_a, t_e\rangle \end{array} \right\}$$

The third and fourth curtailed norms above (counting from top to bottom) have been rendered useless, as they have unsatisfiable constraints—this becomes apparent if we "unravel" the norms, that is,

$$\left\langle \mathsf{F}_{A:R} deploy(S, X, Y) \circ \left\{ \boxed{S = s_1}, X = 100, Y > 50, \boxed{S \neq s_1} \right\}, t_d, t_a, t_e \right\rangle$$

$$\left\langle \mathsf{F}_{A:R} deploy(S, X, Y) \circ \left\{ S = s_1, \boxed{X = 100}, Y > 50, \boxed{X \neq 100} \right\}, t_d, t_a, t_e \right\rangle$$

We enclose in boxes the unsatisfiable constraints. Such norms are not wrong: they have simply been curtailed to an extent that renders them useless. To further ground our approach, we now outline our realisation of the conflict resolution mechanism.

The seemingly superfluous constraints added by transforming norms into the extended explicit unification format allow us to handle norms without constraints. When two norms without constraints are in conflict and one of them is to be curtailed, our mechanism will use the superfluous constraints (of the preserved norm) to render the curtailed norm useless. For example, if we have

$$\mathsf{F}_{A_1:R_1}\, p(X_1) \circ \{A_1 = A', R_1 = R', X_1 = X'\}$$
$$\mathsf{O}_{A_2:R_2}\, p(X_2) \circ \{A_2 = A'', R_2 = R'', X_2 = X''\}$$

such that the former is to be curtailed and the latter is to be preserved, then we get the following set of curtailed prohibitions:

$$\left\{ \begin{array}{l} \mathsf{F}_{A_1:R_1}\, p(X_1) \circ \{A_1 = A', R_1 = R', X_1 = X', \boxed{A_1 \neq A''}\} \\ \mathsf{F}_{A_1:R_1}\, p(X_1) \circ \{A_1 = A', R_1 = R', X_1 = X', \boxed{R_1 \neq R''}\} \\ \mathsf{F}_{A_1:R_1}\, p(X_1) \circ \{A_1 = A', R_1 = R', X_1 = X', \boxed{X_1 \neq X''}\} \end{array} \right\}$$

The boxed constraints are unsatisfiable: they establish that one of the formula's variables cannot have any values as A'', R'' and X'' are all uninstantiated variables. The curtailed norms have been rendered useless as their scope of influence is non-existent. The set of curtailed norms in this case is, in practice, an empty set as the constraints are unsatisfiable.

Although the explicit unifications are required for our conflict resolution mechanism to work properly, they make norms more complex and harder to understand. In our examples, to improve readability we shall use norms represented in the usual format (i.e., with arbitrary terms appearing in them), however they can be automatically translated into their extended explicit unification format.

4.3 An implementation of norm curtailment

We show in Fig. 4 a prototypical implementation of the curtailment mechanism as a logic program. We show our logic program with numbered lines to enable the easy referencing of its constructs. Lines 1–7 define *curtail*, and lines 8–14 define an auxiliary predicate *merge*. Lines 1–6 depict the case when the norms are in conflict: the test in line 4 ensures this. Line 5 invokes the auxiliary predicate *merge*, which, as the name suggests, merges the constraints $\{\gamma_0, \ldots, \gamma_n\}$ with the negated constraints $\neg\gamma'_j$. Line 6 assembles Ω by collecting the members Γ of the list *New*Γ*s* and using them to create curtailed versions of ω. The elements of the list *New*Γ*s* assembled via *merge* are of the form $\Gamma \cup \{(\neg\gamma'_j \cdot \sigma)\}$; additionally, in our implementation, we check if each new set of constraints is satisfiable (line 10). As shown in the previous examples, the merging of constraints may give rise to curtailed norms that are never applicable as their constraints cannot be satisfied; these are discarded in our implementation.

4.4 Curtailment policies

Rather than assuming that a specific deontic modality is always curtailed,[5] we propose to explicitly use *policies* for determining, given a pair of norms, which one is to be curtailed. Such policies confer more flexibility on our curtailment mechanism:

[5] In Vasconcelos *et al.* [41], for instance, prohibitions are always curtailed. This ensures the choices of agents' behaviour are kept as open as possible.

```
1   curtail(ω, ω′, Ω) ←
2     ω = ⟨X_{α:ρ}φ ∘ {γ0, ..., γn}, td, ta, te⟩∧
3     ω′ = ⟨X′_{α′:ρ′}φ′ ∘ {γ′0, ..., γ′m}, t′d, t′a, t′e⟩∧
4     conflict(ω, ω′, σ)∧
5     merge([(¬γ′0 · σ), ..., (¬γ′m · σ)], {γ0, ..., γn}, NewΓs)∧
6     setof(⟨X_{α:ρ}φ ∘ Γ, td, ta, te⟩, member(Γ, NewΓs), Ω)
7   curtail(ω, ω′, {ω})

8   merge([], _, [])
9   merge([(¬γ′ · σ)|Gs], Γ, [NewΓ|MoreΓs]) ←
10      satisfy(Γ ∪ (¬γ′ · σ))∧
11      NewΓ = Γ ∪ {(¬γ′ · σ)}∧
12      merge(Gs, Γ, MoreΓs)
13  merge([_|Gs], Γ, NewΓs) ←
14      merge(Gs, Γ, NewΓs)
```

Fig. 4 Implementation of *curtail* as a logic program

Definition 11 A policy π is a tuple $\langle \omega, \omega', \Gamma \rangle$ establishing that ω should be curtailed (and ω' should be preserved), if constraints Γ hold.

For example, let us consider policy

$$\langle \langle \nu, T_d, T_a, T_e \rangle, \langle \nu', T'_d, T'_a, T'_e \rangle, \{T_d < T'_d\} \rangle$$

This policy states that when two norms, ω and ω', are in conflict, ω is to be curtailed if its time of declaration T_d precedes that of ω'. Adding constraints to policies gives us a fine-grained control over how conflicts should be handled, and, allows us to capture classic forms of deontic conflict resolution, such as *lex posterior* (the most recent norm takes precedence) and *lex superior* (the norm imposed by the strongest power takes precedence) [24]. The policy above formally defines *lex posterior*, establishing a precedence relationship between the two norms (with respect to their declaration times); in Sect. 7 we formalise *lex superior*. We shall represent a set of such policies as Π.

5 Management of normative states

In this section, we give details on how our approach to conflict detection and resolution can be used to manage normative states. In detailing our approach, we explain how we preserve conflict-freedom when adopting a new norm (Sect. 5.1) as well as how norms are removed (Sect. 5.2)—when a norm is removed we must ensure that any curtailment it caused is undone.

5.1 Norm adoption

The algorithm *adoptNorm* shown in Fig. 5 describes how a a norm ω can be added to an existing conflict-free (possibly empty) set Ω_{old} such that conflicts arising from adopting ω are resolved to ensure that the resulting set of norms, Ω_{new}, is conflict free. The algorithm uses a set Π of policies determining how the curtailment of conflicting norms should be carried out.

The norm adoption algorithm employs an auxiliary set of norms Ω_{aux} initially set to store ω, the norm to be inserted in Ω_{old}. The algorithm works by performing the outermost **while** loop as long as the set Ω_{aux} has any elements in it. Both Ω_{aux} and Ω_{old} may change as a

```
algorithm adoptNorm(ω, Ω_old, Π, Ω_new)
input ω, Ω_old, Π
output Ω_new
Ω_aux := {ω}
Ω_new := ∅
while Ω_aux ≠ ∅ do
    Ω'_aux := Ω_aux
    for_each ω_aux ∈ Ω_aux do
        conflict_flag := false
        Ω'_old := Ω_old
        for_each ω_old ∈ Ω_old do
            if conflict(ω_aux, ω_old, σ) then // test for conflict
                conflict_flag := true
                if existPolicy(ω_old, ω_aux, σ, Π) then
                    curtail(ω_old, ω_aux, Ω') // curtail ω_old w.r.t. ω_aux
                    Ω'_old := (Ω'_old − {ω_old}) ∪ Ω' // update Ω'_old
                else
                    curtail(ω_aux, ω_old, Ω') // curtail ω_aux w.r.t. ω_old
                    Ω'_aux := (Ω'_aux − {ω_aux}) ∪ Ω' // update Ω'_aux
                end_if
            end_if
        end_for
        if conflict_flag = false then // if no conflicts
            Ω_new := Ω_new ∪ {ω_aux} // add ω_aux to Ω_new
            Ω'_aux := Ω'_aux − {ω_aux} // remove ω_aux from Ω'_aux
        end_if
    end_for
    Ω_aux := Ω'_aux // update set of norms to be adopted
    Ω_old := Ω'_old // update old set of norms
end_while
Ω_new := Ω_new ∪ Ω_old
```

```
algorithm existPolicy(ω, ω', σ, Π) // auxiliary test to find a policy
input ω, ω', σ, Π
for_each ⟨ω_π, ω'_π, Γ⟩ ∈ Π
    if unify(ω, ω_π, σ) and unify(ω', ω'_π, σ) and satisfy(Γ · σ) then
        return true // return true upon finding a policy
    end_if
end_for
return false // otherwise return false
```

Fig. 5 Norm adoption algorithm (*top*) and check for policy (*bottom*)

result of norms being curtailed, so the algorithm uses Ω'_{aux} and Ω'_{old}, respectively, to update those sets. This is required as both sets control **for_each** loops and we only update these sets after the loops are completed—we do so immediately before the **end_while** construct in the algorithm.

Within the **while** loop, the algorithm has two nested **for_each** loops. The outermost **for_each** loop goes through each norm $\omega_{aux} \in \Omega_{aux}$ and checks for conflicts with all old norms $\omega_{old} \in \Omega_{old}$. Although initially Ω_{aux} is a singleton set $\{\omega\}$, ω may be curtailed, thus giving rise to a set of curtailed norms which, in their turn, must be checked for conflicts with the existing (old) norms; hence the need for the set Ω_{aux}.

Within the innermost **for_each** loop each pair $\omega_{aux}, \omega_{old}$ is checked for conflicts via $conflict(\omega_{aux}, \omega_{old}, \sigma)$. If this is the case, then the algorithm initially checks (in the second **if** construct), using the auxiliary *existPolicy* test (explained below), if there is a policy in Π establishing that ω_{old} should be curtailed and ω_{aux} should be preserved. If such a policy

cannot be found, then the algorithm, by default, curtails ω_{aux}—this is the rationale for the first **else** construct (within the innermost loop).

When ω_{old} is curtailed (second **if** construct), then the algorithm updates the "working" set of old norms Ω'_{old}, removing ω_{old} from it and adding Ω', the curtailed versions of ω_{old}. Likewise, in the **else** of the second **if** construct, when ω_{aux} is curtailed the algorithm updates the working set of new norms Ω'_{aux}, removing ω_{aux} from it and adding Ω', the curtailed versions of ω_{aux}.

The algorithm keeps track of those norms $\omega_{aux} \in \Omega_{aux}$ which are not in conflict with any of the norms $\omega_{old} \in \Omega_{old}$. This is done via the boolean variable *conflict_flag*, initialised to **false** at the start of the outermost **for_each** loop. If ω_{aux} is not in conflict with any ω_{old}, then ω_{aux} is added to the set Ω_{new} and it is removed from the set Ω'_{aux}. Set Ω'_{aux} is used to update the set Ω_{aux} of norms to be adopted. The norms in Ω_{aux} which are in conflict with norms in Ω_{old} either cause norms in Ω_{old} to get curtailed (second **if** construct) or are curtailed themselves (the **else** of the second **if** construct), becoming free of conflicts with any norm in Ω_{old}. The set Ω_{old} will eventually become empty thus ensuring the algorithm always terminates.

The set Ω_{new} is gradually built by adding norms ω_{aux} which do not conflict with any norm ω_{old}; then Ω_{old} is added to Ω_{new} (last line of algorithm). This is the correct outcome in that the set Ω_{new} is conflict-free and has all norms of Ω_{old} (possibly curtailed) and all norms of Ω_{aux} (also possibly curtailed; it should be noted that Ω_{aux} initially contains ω, the norm to be adopted). The norms of Ω_{old} either get preserved in Ω'_{old} (if not in conflict) or they are replaced by their curtailments (cf. second **if** construct). Ω'_{old} is used to update Ω_{old} (cf. line immediately before **end_while**) which, on its turn is added to Ω_{new} (last line of algorithm). Ω_{new} is guaranteed to be conflict-free: only ω_{aux} which are not in conflict with any ω_{old} are added to Ω_{new}. The Ω_{old} norms added to Ω_{new} at the end of the **while** loop are also conflict-free and do not conflict with any norms in Ω_{aux} (otherwise Ω_{aux} would not be empty and the loop would not have come to an end). The algorithm, however, will only work if Ω_{old} is conflict-free.

The *adoptNorm* algorithm makes use of an auxiliary test *existPolicy* shown on the bottom of Fig. 5—given two norms ω, ω', a substitution σ and a set of policies Π, *existPolicy* returns **true** if there is a policy in Π stipulating that ω is to be curtailed with respect to ω'. When checking for a policy that is applicable, the algorithm uses unification to check (i) whether ω matches/unifies with ω_π and ω' with ω'_π; and (ii) whether the policy constraints hold under the given σ.

The space complexity of *adoptNorm* is, however, exponential in the worst case. To show this, let us assume the auxiliary set Ω_{aux} has an initial norm ω which is in conflict with all n norms in Ω_{old}; to simplify our analysis, let us further assume each norm in Ω_{old} has the same number c of constraints and that each constraint (in its negated form) will give rise to a curtailment of ω. When we compare ω with the first norm in Ω_{old} we obtain c curtailed versions of ω (one for each constraint c of the first norm); these curtailed norms are, in their turn, checked for conflicts with the second norm in Ω_{old}, yielding $c \times c$ norms, and so on, for each of the n norms in Ω_{old}. We shall thus have $(c \times \cdots \times c)$ n times, that is, c^n. Since Ω_{aux} is used to control the **while** loop, the space complexity will directly impact on the time complexity which will be, in the worst case, also exponential.

Another possible extreme case arises when all norms in Ω_{old} get curtailed, thus increasing the size of that set. If we assume Ω_{aux} has an initial norm ω which is in conflict with all n norms in Ω_{old}, and if we further assume that the policies in place will preserve ω and curtail the norms in Ω_{old}, and that ω has c constraints, we shall end up with c new norms for each norm in Ω_{old}, that is, $n \times c$. It is worth pointing out that the new norms in Ω_{old} will

never conflict with each other (assuming Ω_{old} was conflict-free to begin with), as any added constraints will only further restrict their scope of influence.

The exponential complexity of *adoptNorm* is due to the growth of curtailed norms in Ω_{aux} and the need to exhaustively check these with the norms in Ω_{old}. This complexity is due to the centralised nature of our solution. A single global normative state is much simpler to manage, but the computational costs can be prohibitively high, as our algorithm above illustrates. An alternative explored in [13] proposes the *distributed* management of normative states: the proposal addressed a class of multi-agent systems whose agents' actions/interactions are broken down into independent (possibly simultaneous) *activities*. Activities are, for instance, the registration of agents with a trusted party, or the enactment of an auction. This distributed solution would allow for various partial normative states to be managed in parallel; these normative states will necessarily be smaller than a single global state, thus cutting down the number of norms each state will have.

Although we carried out a worst-case analysis, in practice it is unlikely that a norm to be adopted will conflict with all other existing norms; it also unlikely that the number of constraints on each individual existing norm will be too high. Notwithstanding these more pragmatic considerations, we can also set an upper bound on the size of Ω_{aux} which can be dealt with, that is, an upper bound on how big the worst case can be, depending on the computational resources available.

The *adoptNorm* algorithm can be made more efficient by "stripping" Ω_{old} of those norms which do not (any more) conflict with any of the norms of Ω_{aux}. The rationale for this is as follows: if an ω_{old} does not conflict with any ω_{aux}, it will not conflict with any of its possible curtailed versions, since these will necessarily have a more limited scope of influence. This stripping process could either be done at the beginning or at the end of the outermost **for_each** loop.

5.2 Norm removal

As well as adding norms to normative states we also need to support their removal. Since the introduction of a norm may have interfered with other norms, resulting in their curtailment, when that norm is removed we must *undo* the curtailments it caused, that is, we must return (or "roll back") to a previous form of the normative state. In order to allow curtailments of norms to be undone, we record the complete *history* of normative states representing the evolution of normative positions of agents:

Definition 12 $\mathcal{H}$ is a non-empty and finite sequence of tuples $\langle \Omega, \omega, \Pi \rangle$, where Ω is a normative state, ω is a norm and Π is a set of policies.

We shall denote the empty history as $\langle\,\rangle$. We define the concatenation of sequences as follows: if $\mathcal{H}$ is a sequence and h is a tuple, then $\mathcal{H} \bullet h$ is a new sequence consisting of $\mathcal{H}$ followed by h. Any non-empty sequence $\mathcal{H}$ can be decomposed as $\mathcal{H} = \mathcal{H}' \bullet h \bullet \mathcal{H}''$, $\mathcal{H}'$ and/or $\mathcal{H}''$ possibly empty. The following properties hold for our histories $\mathcal{H}$:

(1) $\mathcal{H} = \langle \emptyset, \omega, \Pi \rangle \bullet \mathcal{H}'$
(2) $adoptNorm(\omega_i, \Omega_i, \Pi, \Omega_{i+1})$

The first condition establishes that the first element of a history has an empty Ω. The second condition establishes the relationship between any two consecutive tuples in histories: normative state Ω_{i+1} is obtained by adding ω_i to Ω_i making use of the set of policies Π.

$\mathcal{H}$ is required to allow the removal of a norm in an orderly fashion, as not only the norm itself has to be removed but also all the curtailments it caused when it was introduced in Ω. $\mathcal{H}$

algorithm $removeNorm(\omega, \mathcal{H}, \Omega, \mathcal{H}')$
input $\omega, \mathcal{H}$
output $\Omega, \mathcal{H}'$
begin
  if $\mathcal{H} = \mathcal{H}' \bullet \langle \Omega_k, \omega, \Pi \rangle \bullet \cdots \bullet \langle \Omega_n, \omega_n, \Pi \rangle$ **then**
  begin
    $\Omega := \Omega_k$
    for $i = k + 1$ **to** n **do**
    begin
      $adoptNorm(\omega_i, \Omega, \Pi, \Omega')$
      $\mathcal{H}' := \mathcal{H}' \bullet \langle \Omega, \omega_i, \Pi \rangle$
      $\Omega := \Omega'$
    end
  end
  else
  begin
    $\mathcal{H} = \mathcal{H}'' \bullet \langle \Omega_n, \omega_n, \Pi \rangle$
    $\Omega := \Omega_n$, $\mathcal{H}' := \mathcal{H}$
  end
end

Fig. 6 Algorithm to remove norms

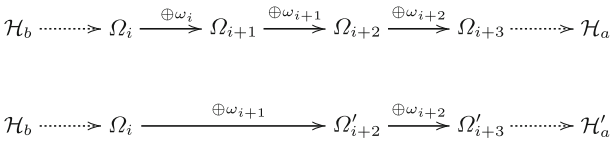

Fig. 7 Histories before (*top*) and after (*bottom*) a norm removal

contains a tuple $\langle \Omega, \omega, \Pi \rangle$ that indicates the introduction of norm ω and, therefore, provides us with a normative state Ω *before* the introduction of ω. The effect of introducing ω can be reversed by using Ω and redoing (performing a kind of "roll forward") all the inclusions of norms according to the sequence represented in $\mathcal{H}$ via *adoptNorm*.

This mechanism is detailed in Fig. 6: algorithm *removeNorm* describes how to remove a norm ω given a history $\mathcal{H}$; it outputs a normative state Ω and an updated history $\mathcal{H}'$ and works as follows. Initially, the algorithm checks if ω indeed appears in $\mathcal{H}$—it does so by matching $\mathcal{H}$ against a pattern of a sequence in which ω appears as part of a tuple (notice that the pattern initialises the new history $\mathcal{H}'$). If there is such a tuple in $\mathcal{H}$, then we initialise Ω as Ω_k, that is, the normative state *before* ω was introduced. Following that, the **for** loop implements a *roll forward*, whereby new normative states (and associated history $\mathcal{H}'$) are computed by introducing the $\omega_i, k + 1 \leq i \leq n$, which come after ω in the original history $\mathcal{H}$. If ω does not occur in any of the tuples of $\mathcal{H}$ (this case is catered by the **else** of the **if** construct) then the algorithm uses pattern-matching to decompose the input history $\mathcal{H}$ and obtain its last tuple—this is necessary as this tuple contains the most recent normative state Ω_n which is assigned to Ω; the new history $\mathcal{H}'$ is the same as $\mathcal{H}$.

Figure 7 illustrates how the mechanism works. It shows two histories $\mathcal{H}$ (top) and $\mathcal{H}'$ (bottom) obtained from $\mathcal{H}$ by removing ω_i; sub-sequences of the full histories are denoted generically as $\mathcal{H}_b$ (that is, the history before the sub-sequence of interest) and $\mathcal{H}_a$ (that is, the history after the sub-sequence of interest). We use "$\oplus$" to denote a norm being introduced via our norm adoption algorithm, with curtailments possibly taking place. The illustration

shows a roll back to Ω_i (i.e., the normative state *before* the adoption of ω_i), and the *re-computation* of all norm adoptions from Ω_i onwards, skipping ω_i (i.e., the roll forward), that is, $adoptNorm(\omega_{i+1}, \Omega_i, \Pi, \Omega'_{i+2})$ and $adoptNorm(\omega_{i+k}, \Omega'_{i+k}, \Pi, \Omega'_{i+k+1}), k > 1$.

The computational costs of obtaining a new normative state Ω (and a corresponding new history $\mathcal{H}'$) using algorithm *removeNorm* is, in the worst case, exponential. The worst case arises when the norm to be removed is the first one which was adopted, that is, we have history

$$\mathcal{H} = \langle\langle\emptyset, \omega_0, \Pi\rangle, \langle\Omega_1, \omega_1, \Pi\rangle, \langle\Omega_2, \omega_2, \Pi\rangle, \ldots, \langle\Omega_{n-1}, \omega_{n-1}, \Pi\rangle, \langle\Omega_n, \omega_n, \Pi\rangle\rangle$$

And we need to remove norm ω_0. In this case, we need to compute the new history

$$\mathcal{H}' = \langle\langle\emptyset, \omega_1, \Pi\rangle, \langle\Omega'_2, \omega_2, \Pi\rangle, \ldots, \langle\Omega'_{n-1}, \omega_{n-1}, \Pi\rangle, \langle\Omega'_n, \omega_n, \Pi\rangle\rangle$$

We also need to obtain the new normative state Ω' resulting from adding ω_n to Ω'_n, that is, $adoptNorm(\omega_n, \Omega'_n, \Pi, \Omega')$. In the worst case, the adoption of ω_i into Ω'_i with m elements will create $c \times m$ curtailed norms, where c is the number of constraints of ω_i (assuming ω_i is in conflict with all norms in Ω'_i). For simplicity, let us assume that all $\omega_i, 1 \leq i \leq n$, have the same number of constraints c. The total cost for computing $\mathcal{H}'$ and Ω' can be equated with the number of norms of all $\Omega'_i, 1 \leq i \leq n$, as each norm in Ω'_i must be checked for conflicts with ω_i, the next norm to be adopted. The number of norms is $|\emptyset| + |\Omega'_2| + |\Omega'_3| + \cdots + |\Omega'_n| = 0 + c + (c \times c) + (c \times c^2) + \cdots + (c \times c^{n-1}) = O(c^n)$.

Again, this complexity is due to the centralised nature of our solution, and here we could also contemplate a distributed approach as previously discussed for the *adoptNorm* algorithm. The exponential complexity of centralised norm management can also be dealt with by restricting norm removal to the last m norms.

6 Indirect conflicts

In our previous discussion, norm conflicts were detected via a direct comparison of atomic formulae representing actions. However, conflicts may also arise *indirectly* via relationships among actions. For instance, let us suppose the following norms are in place:

$$\mathsf{P}_{A:R}\, move(X, Y) \qquad \mathsf{F}_{A:R}\, fly(X, Y)$$

They represent, respectively, that any agent A in any role R is permitted to *move* from X to Y, and any agent A in any role R is forbidden to *fly* from X to Y. Let us further suppose that the only way to move around a particular scenario is flying, that is, $move(X, Y)$ amounts to $fly(X, Y)$. In this case, we can rewrite the norms above as

$$\mathsf{P}_{A:R}\, fly(X, Y) \qquad \mathsf{F}_{A:R}\, fly(X, Y)$$

which are in conflict. We can thus say that an *indirect* conflict arises between the original norms. In addition to indirect conflicts caused by relationships among actions, we also formally capture a class of indirect conflicts due to delegation of tasks (and norms) among agents. These conflicts are presented in the subsections below.

6.1 Indirect conflicts due to relationships among actions

For this kind of indirect conflicts, we make use of a set of *domain axioms* in order to declare domain-specific relationships between actions:

Definition 13 The set of domain axioms Δ is a finite and possibly empty set of formulae $\varphi \rightarrow (\varphi'_1 \wedge \cdots \wedge \varphi'_n)$ where $\varphi, \varphi'_i, 1 \leq i \leq n$, are atomic first-order formulae.

The domain axioms formally forge inter-relationships among actions—these dependencies could be an action "break-down" into sub-actions, an action's side-effects as well as its causal effects. For instance, the action "clear up oil spill" will break down into "stop oil leakage", "contain oil spill" and "remove oil". This formalisation is aimed at capturing a kind of "counts as" relationship [34] among actions: $\varphi \rightarrow (\varphi'_1 \wedge \cdots \wedge \varphi'_n)$ means that φ *counts as* $(\varphi'_1 \wedge \cdots \wedge \varphi'_n)$, and we do not differentiate the various dependencies (i.e., breakdown of a composite action, side effects and causal effects) which might exist among actions. We assume that a set of domain axioms is available to us, but acknowledge that in realistic scenarios the preparation of such a set is not a trivial task, involving careful knowledge elicitation [11].

In order to address indirect conflicts between norms based on domain-specific relationships between actions, we have to adapt our curtailment mechanism. With the introduction of domain axioms, the conflict check has to be performed for each of the conjuncts in this relationship. For example, let us suppose we have the following set of domain axioms

$$\Delta = \left\{ evacuate(A, B, X, Y) \longrightarrow \begin{pmatrix} gather\_intelligence(A, X, Y) \wedge \\ fly(A, helicopter, X, Y) \wedge \\ uplift(A, helicopter, B) \end{pmatrix} \right\}$$

where $evacuate(A, B, X, Y)$ represents that A is to evacuate B from area (X, Y). Let us also suppose we have norm

$$\langle \mathsf{P}_{A:R} evacuate(a, b, X, Y) \circ \{X > 50\}, t_d, t_a, t_e \rangle.$$

In this case, actions

$$gather\_intelligence(a, X, Y) \circ \{X > 50\}$$
$$fly(a, helicopter, X, Y) \circ \{X > 50\}$$
$$uplift(a, helicopter, b)$$

are also permitted. We revisit Def. 9, extending it to address indirect conflicts:

Definition 14 A conflict arises between ω and ω' under a set of domain axioms Δ and a substitution σ, denoted as $conflict^*(\Delta, \omega, \omega', \sigma)$, iff:

(1) $conflict(\omega, \omega', \sigma)$, or
(2) $\omega = \langle \mathsf{X}_{\alpha:\rho} \varphi \circ \Gamma, t_d, t_a, t_e \rangle$, there is a domain axiom $(\varphi' \rightarrow (\varphi'_1 \wedge \cdots \wedge \varphi'_m)) \in \Delta$ such that $unify(\varphi, \varphi', \sigma')$, and $\bigvee_{i=1}^{m} conflict^*(\Delta, \langle \mathsf{X}_{\alpha:\rho} \varphi'_i \circ \Gamma, t_d, t_a, t_e \rangle \cdot \sigma', \omega', \sigma)$

The above definition recursively follows a chain of indirect conflicts, looking for any two conflicting norms. Case 1 provides the base case of the recursion, checking if norms ω, ω' are in direct conflict. Case 2 addresses the general recursive case: if a norm X (that is, O, P or F) on an action φ unifies with φ' on the left-hand side of a domain axiom $(\varphi \rightarrow (\varphi'_1 \wedge \cdots \wedge \varphi'_m)) \in \Delta$, then we "transfer" the norm from φ to $\varphi'_1, \ldots, \varphi'_m$, thus obtaining $\langle \mathsf{X}_{\alpha:\rho} \varphi'_i \circ \Gamma, t_d, t_a, t_e \rangle$, $1 \leq i \leq m$. If we (recursively) find an indirect conflict between ω' and at least one of these norms, then an indirect conflict arises between the original norms ω and ω'. It is important to notice that the substitution σ' that unifies φ and φ' is factored in the mechanism: we apply it to the new φ'_i formulae in the recursive call(s). The resulting σ will contain a chain of substitutions which will allow the indirect conflict between ω and ω' to be resolved via the norm curtailing mechanism.

6.2 Indirect conflicts due to delegation among agents

In this subsection, we formally capture another kind of indirect conflict, viz., the one caused when agents delegate norms across the community. Initially, we introduce the following special logical operator:

$$\varphi \xrightarrow[\alpha:\rho \quad \alpha':\rho']{} (\varphi'_1 \wedge \cdots \wedge \varphi'_m) \tag{1}$$

Such formulae are named *delegation axioms* and they represent that agent α adopting role ρ can transfer any norms on action φ to agent α' adopting role ρ'; however, the norm will be transfered to actions $\varphi'_1 \wedge \cdots \wedge \varphi'_m$ instead. The semantics of this operator is as follows:

- if $\omega = \langle \mathsf{X}_{\alpha:\rho}\varphi \circ \Gamma, t_d, t_a, t_e\rangle \in \Omega$, and
- we have $(\varphi' \xrightarrow[\alpha':\rho' \quad \alpha'':\rho'']{} (\varphi'_1 \wedge \cdots \wedge \varphi'_m))$ such that $unify(\langle \varphi, \alpha, \rho\rangle, \langle \varphi', \alpha', \rho'\rangle, \sigma)$, and $satisfy(\Gamma \cdot \sigma)$

then $(\langle \mathsf{X}_{\alpha'':\rho''}\varphi'_i \circ \Gamma, t_d, t_a, t_e\rangle \cdot \sigma) \in \Omega$, $1 \leq i \leq m$.

Actions/norms are delegated following a pre-established relationship forged among the roles adopted by the agents. This relationship may represent, for instance, a simple team of peers, a hierarchy, or any arbitrary agent organisation [27]. We formally represent the inter-role relationship via a partial ordering $\succ$ among roles: $r \succ r', r, r' \in Roles$ means that agents adopting role r can delegate actions/norms to those agents adopting role r'. This formalisation can be seen as a simple notion of power [9,12].

Delegation and domain axioms are both stored in Δ. However, for all delegation axioms in Δ of the form of formula 1, it is the case that $\rho \succ \rho'$, that is, only delegation axioms which respect the inter-role relationships are kept in Δ. We extend Def. 14 to accommodate delegation axioms:

Definition 15 A conflict arises between two norms ω and ω' under a set of domain/delegation axioms Δ, denoted as $conflict^*(\Delta, \omega, \omega', \sigma)$, iff:

(1) $conflict(\omega, \omega', \sigma)$, or

(2) $\omega = \langle \mathsf{X}_{\alpha:\rho}\varphi \circ \Gamma, t_d, t_a, t_e\rangle$, there is a domain axiom $(\varphi' \rightarrow (\varphi'_1 \wedge \cdots \wedge \varphi'_m)) \in \Delta$ such that $unify(\varphi, \varphi', \sigma')$, and $\bigvee_{i=1}^{m} conflict^*(\Delta, \langle \mathsf{X}_{\alpha:\rho}\varphi'_i \circ \Gamma, t_d, t_a, t_e\rangle \cdot \sigma', \omega', \sigma)$, or

(3) $\omega = \langle \mathsf{X}_{\alpha:\rho}\varphi \circ \Gamma, t_d, t_a, t_e\rangle$, there is a delegation axiom $(\varphi' \xrightarrow[\alpha':\rho' \quad \alpha'':\rho'']{} (\varphi'_1 \wedge \cdots \wedge \varphi'_m)) \in \Delta$ such that $unify(\langle \varphi, \alpha, \rho\rangle, \langle \varphi', \alpha', \rho'\rangle, \sigma')$, $satisfy(\Gamma \cdot \sigma')$, and $\bigvee_{i=1}^{m} conflict^*(\Delta, \langle \mathsf{X}_{\alpha'':\rho''}\varphi'_i \circ \Gamma, t_d, t_a, t_e\rangle \cdot \sigma', \omega', \sigma)$

Cases 1 and 2 are as before. In case 3, we obtain a delegation axiom and check if its action, role and agent unify with those of ω. The norm will be transferred to the new actions $(\varphi'_1 \wedge \cdots \wedge \varphi'_m)$ but these will be associated with a possibly different agent/role pair $\alpha'':\rho''$. The new norms are recursively checked and if at least one of them conflicts with ω', then an indirect conflict caused by delegation arises. Means to detect loops in delegation must be added to the definition above; in McCallum et al. [27] one finds an exploration of this and other issues concerning delegation and influence in agent organisations.

7 Authority associated with norms

In this section, we propose a simple means to represent *authority*, which allows us to capture interesting phenomena. By "authority" we mean agents to whom the norm holder is responsible for complying with the norm. We focus on authority given by the agent's position, that is, the one conferred on an agent by virtue of its adopted role within a society.

7.1 A formal representation of authority

We add to our norms the representation of the authority to whom the norm holder is responsible. Since agents may have distinct powers depending on the roles they adopt, we represent authority also as an agent/role pair. We extend our deontic modalities with a superscript as in, for instance, $\mathsf{O}^{\alpha:\rho}_{\alpha':\rho'}\varphi \circ \Gamma$, representing that agent $\alpha':\rho'$ is obliged to do $\varphi \circ \Gamma$, and $\alpha:\rho$ is the authority to whom $\alpha':\rho'$ is to be held responsible for complying with the norm. We re-use the inter-role relationship of Subsect. 6.2 and require the authority annotations to comply with them, that is, $\mathsf{O}^{\alpha:\rho}_{\alpha':\rho'}\varphi \circ \Gamma$ can only hold if $\rho \succ \rho'$.

This explicit representation of authority allows us to define a useful class of curtailment policies. These policies help to solve normative conflicts by curtailing those norms in conflict with a less powerful authority. This is the principle of *lex superior* discussed in [24]: the norm with the most powerful associated authority should be the one preserved, if other norms are in conflict. A generic formulation for the *lex superior* policy using our representation of authority is thus:

$$\langle \langle \mathsf{X}^{\alpha_2:\rho_2}_{\alpha_1:\rho_1}\varphi \circ \Gamma, T_d, T_a, T_e\rangle, \langle \mathsf{X}'^{\alpha'_2:\rho'_2}_{\alpha'_1:\rho'_1}\varphi' \circ \Gamma', T'_d, T'_a, T'_e\rangle, \{\rho'_2 \succ \rho_2\}\rangle$$

where X stands for either O, P or F. The policy establishes that norm $\langle \mathsf{X}^{\alpha_2:\rho_2}_{\alpha_1:\rho_1}\varphi \circ \Gamma, T_d, T_a, T_e\rangle$ is to be curtailed, and $\langle \mathsf{X}'^{\alpha'_2:\rho'_2}_{\alpha'_1:\rho'_1}\varphi' \circ \Gamma', T'_d, T'_a, T'_e\rangle$ is to be preserved, if the role ρ'_2 of the authority associated with the latter norm is more powerful than the role ρ_2 of the authority associated with the former norm.

7.2 A generalised notion of delegation

The explicit representation of authority introduced above allows us to formalise a generalised notion of norm delegation. We extend the logical operator to capture delegations introduced in Subsect. 6.2, adding authority annotations:

$$\varphi \xrightarrow[\alpha_1:\rho_1 \quad \alpha_2:\rho_2]{\alpha_3:\rho_3 \quad \alpha_4:\rho_4} \varphi'_1 \wedge \cdots \wedge \varphi'_n \tag{2}$$

The construct means that "a norm on action φ assigned to agent $\alpha_1 : \rho_1$ with associated authority $\alpha_3 : \rho_3$ can be transfered to a norm on actions $\varphi'_1 \wedge \cdots \wedge \varphi'_n$ to be assigned to agent $\alpha_2 : \rho_2$ who will be held responsible to authority $\alpha_4 : \rho_4$". An example of this operator is

$$\mathit{reconnaissance}(X, Y) \xrightarrow[A_2:R_2 \quad A_3:R_3]{A_1:R_1 \quad A_1:R_1} \mathit{gather\_intelligence}(X, Y) \tag{3}$$

In this formula, the authority associated with the norm is *preserved* in the delegation: agent $A_3 : R_3$, to whom the norm is delegated, will be held responsible to the same authority $A_1:R_1$ of the original norm. This phenomenon is named *delegation with preservation of authority*.

Another example of the generalised delegation operator is

$$\mathit{gather\_intelligence}(X, Y) \xrightarrow[A_2:R_2 \quad A_3:R_3]{A_1:R_1 \quad A_2:R_2} \mathit{interview}(X, Y) \tag{4}$$

In this formula, the authority associated with the norm is *changed* in the delegation: agent $A_3 : R_3$ to whom the norm is delegated will now be held responsible for abiding by the norm to agent $A_2 : R_2$, that is, the agent to whom the norm was originally assigned, and *not* agent $A_1 : R_1$, the original authority. This phenomenon is named *delegation of authority*.

Formulae such as 3 and 4 are called *authority axioms* and they establish conditions under which norms can be delegated, stating to whom they are to be delegated as well as with which authority the delegated norms are to become associated. For instance, let us consider the following norm

$$\mathsf{O}^{a_1:cmd}_{a_2:ltn} reconnaissance(20, W) \circ \{W > 100\}$$

It states that agent $a_1 : cmd$ is the authority to whom $a_2 : ltn$ is responsible for complying with the obligation to do a reconnaissance of area $(20, W)$, $W > 100$. Let us further assume in our scenario the following power relationship for the set of roles $Roles = \{sld, srg, ltn, cmd\}$:

$$cmd \succ ltn, ltn \succ srg, srg \succ sld$$

That is, *cmd* (for *commander*) is more powerful than *ltn* (for *lieutenant*), lieutenant is more powerful than *srg* (for *sergeant*), and sergeant is more powerful than *sld* (for *soldier*). In this scenario we can apply axiom 3 to obtain the following (delegated) norm

$$\mathsf{O}^{a_1:cmd}_{A_3:srg} gather\_intelligence(20, W) \circ \{W > 100\}$$

We can see that the authority has been preserved, whereas the norm has been delegated from lieutenant a_2 to a sergeant. This new norm can, in its turn, be used in conjunction with axiom 4 to obtain the following norm:

$$\mathsf{O}^{A_3:srg}_{A'_3:sld} interview(20, W) \circ \{W > 100\}$$

That is, the authority associated with the norm is now a sergeant, and the norm has been delegated to a soldier.

To increase the expressiveness of our logical operator, we allow any of the four annotations of the arrow to be left out. If the left-hand side annotations are omitted then they are considered as fresh variables and they will match with any terms actual formulae may have. If we omit the right-hand annotations, then it means that the original subscript and superscript of the formulae matching the left-hand side of the axiom will be preserved in the new formulae of the right-hand side. Thus the machinery for indirect conflicts and delegation axioms explained in Sect. 6 can be seen as a special case of how authority axioms are dealt with.

7.3 Norm violation and sanctions by authority

Our representation of authority allows us to provide an account of norm violation and its associated sanctions. As discussed in Sect. 3.3, a norm violation comes about, for instance, when an action in the global enactment state $\langle a : r, \bar{\varphi}, t\rangle \in \Xi$ is within the scope of a prohibition $\langle \mathsf{F}^{\alpha_1:\rho_1}_{\alpha_2:\rho_2}\varphi \circ \Gamma, t_d, t_a, t_e\rangle \in \Omega$ in the global normative state, that is, $inScope(\langle a : r, \bar{\varphi}, t\rangle, \langle \mathsf{F}^{\alpha_1:\rho_1}_{\alpha_2:\rho_2}\varphi \circ \Gamma, t_d, t_a, t_e\rangle)$ (cf. definition in Fig. 1)—a prohibited action has been carried out. In this case, authority $\alpha_1 : \rho_1$ can impose sanctions or punishments on $\alpha_2 : \rho_2$.

Following our work reported in [15], we represent sanctions/punishments as actions to be added to the global enactment state—these can be, for instance, issuing a fine to the offending agent. A formal representation of this norm violation and its sanction is:

$$\left(\begin{array}{c} \langle a : r, \bar{\varphi}, t\rangle \in \Xi \wedge \\ \langle \mathsf{F}^{\alpha_1:\rho_1}_{\alpha_2:\rho_2}\varphi \circ \Gamma, t_d, t_a, t_e\rangle \in \Omega \wedge \\ inScope(\langle a : r, \bar{\varphi}, t\rangle, \langle \mathsf{F}^{\alpha_1:\rho_1}_{\alpha_2:\rho_2}\varphi \circ \Gamma, t_d, t_a, t_e\rangle) \end{array}\right) \rightsquigarrow \oplus\langle \alpha : \rho, fine(\alpha_2, \rho_2, m), t + 1\rangle$$

8 A formal representation of our scenario

In this section, we revisit the scenario described informally in Sect. 2 and represent some of its aspects with a view to explore our conflict resolution mechanisms. In our formalisation, we omit the superscript annotation denoting authority, as this aspect is not explored; additionally, to improve clarity, we present norms in their usual format, and not with extended sets of constraints and explicit unifications. The initial set of norms for team A in our scenario is

$$\Omega_a = \left\{ \begin{array}{c} \langle \mathsf{O}_{a_1:cmd} evacuate(a_1, workers, 40, 50), 1, 1, +\infty \rangle, \\ \langle \mathsf{F}_{A:R} share(Info, Ag), 1, 1, +\infty \rangle, \\ \langle \mathsf{F}_{A:R} fly(A, helicopter, X, Y), 1, 1, +\infty \rangle \end{array} \right\}$$

where $evacuate(A, B, X, Y)$ represents that A is to evacuate B from area (X, Y) (we assume a two-dimensional coordinate space and that the stranded workers are in area $(40, 50)$ of this grid), $share(Info, Ag)$ represents that $Info$ is to be shared with Ag, and $fly(B, C, X, Y)$ represents that B is to fly C in area (X, Y). In our discussion, all norms above have infinite expiry times.

The prohibition to fly helicopters is in place because in our scenario we have bad weather. The following domain axiom relates the high-level *evacuate* action with obtaining intelligence about the area, flying a helicopter to the area and using it to uplift people:

$$evacuate(A, B, X, Y) \longrightarrow \left(\begin{array}{c} gather\_intelligence(A, X, Y) \wedge \\ fly(A, helicopter, X, Y) \wedge \\ uplift(A, helicopter, B) \end{array} \right)$$

The domain axiom above provides us with the following norms:

$$\Omega_a^1 = \left\{ \begin{array}{c} \langle \mathsf{O}_{a_1:cmd} gather\_intelligence(a_1, 40, 50), 1, 1, +\infty \rangle, \\ \langle \mathsf{O}_{a_1:cmd} fly(a_1, helicopter, 40, 50), 1, 1, +\infty \rangle, \\ \langle \mathsf{O}_{a_1:cmd} uplift(a_1, helicopter, workers), 1, 1, +\infty \rangle, \\ \langle \mathsf{F}_{A:R} share(Info, Ag), 1, 1, +\infty \rangle, \\ \langle \mathsf{F}_{A:R} fly(A, helicopter, X, Y), 1, 1, +\infty \rangle \end{array} \right\}$$

A conflict is detected in the set Ω_a^1: helicopters are simultaneously forbidden and obliged to fly. The prohibition to fly extends to anywhere, as there is bad weather forecast where the helicopters are. The obligation to fly, however, is for a particular area, viz., $(40, 50)$.

The commander of team A may consider asking team B for help: since team B helicopters are not affected by the bad weather, they could be flown into the area to uplift the workers. However, team B needs information about the area helicopters are to be flown into. Team A thus must add to its set of norms a permission to share information on area $(40, 50)$ with a contact agent ag_2 from team B, and we have the following set of norms:

$$\Omega_a^2 = \left\{ \begin{array}{c} \langle \mathsf{O}_{a_1:cmd} evacuate(a_1, workers, 40, 50), 1, 1, +\infty \rangle, \\ \langle \mathsf{F}_{A:R} share(Info, Ag), 1, 1, +\infty \rangle, \\ \langle \mathsf{F}_{A:R} fly(B, helicopter, X, Y), 1, 1, +\infty \rangle, \\ \langle \mathsf{P}_{A:R} share(info(40, 50), ag_2), 1, 1, +\infty \rangle \end{array} \right\}$$

The prohibition to share any information is in conflict with the permission to share specific information on region $(40, 50)$.

Team A can use the conflict resolution machinery to explore alternatives. Norms Ω_a^1 can have its conflict resolved as follows:

$$\Omega_a^3 = \left\{ \begin{array}{c} \langle \mathsf{O}_{a_1:cmd} gather\_intelligence(a_1, 40, 50), 1, 1, +\infty \rangle, \\ \langle \mathsf{O}_{a_1:cmd} fly(a_1, helicopter, 40, 50), 1, 1, +\infty \rangle, \\ \langle \mathsf{O}_{a_1:cmd} uplift(a_1, helicopter, workers), 1, 1, +\infty \rangle, \\ \langle \mathsf{F}_{A:R} share(Info, Ag), 1, 1, +\infty \rangle, \\ \langle \mathsf{F}_{A:R} fly(A, helicopter, X, Y) \circ \{A \neq ag_1\}, 1, 1, +\infty \rangle, \\ \langle \mathsf{F}_{A:R} fly(A, helicopter, X, Y) \circ \{X \neq 40\}), 1, 1, +\infty \rangle, \\ \langle \mathsf{F}_{A:R} fly(A, helicopter, X, Y) \circ \{Y \neq 50\}), 1, 1, +\infty \rangle \end{array} \right\}$$

That is, the prohibition has three "exceptions", which accommodate the obligation to fly the helicopter: it is acceptable to fly the helicopter as long as the agent is ag_1, and the region is $(40, 50)$. Likewise, norms Ω_a^2 can have its conflicts resolved as below:

$$\Omega_a^4 = \left\{ \begin{array}{c} \langle \mathsf{O}_{a_1:cmd} evacuate(a_1, workers, 40, 50), 1, 1, +\infty \rangle, \\ \langle \mathsf{F}_{A:R} share(Info, Ag) \circ \{Info \neq info(40, 50)\}, 1, 1, +\infty \rangle, \\ \langle \mathsf{F}_{A:R} share(Info, Ag) \circ \{Ag \neq ag_2\}, 1, 1, +\infty \rangle, \\ \langle \mathsf{F}_{A:R} fly(B, helicopter, X, Y), 1, 1, +\infty \rangle, \\ \langle \mathsf{P}_{A:R} share(info(40, 50), ag_2), 1, 1, +\infty \rangle \end{array} \right\}$$

That is, the prohibition accommodates two exceptions: it is acceptable to share information as long as the agent with which to share information is ag_2 and the shared information is $info(40, 50)$.

Team A has information to decide on a course of action, that is, if it is preferable to fly helicopters in bad weather or to share intelligence with a coalition partner. Our conflict detection and resolution mechanisms support this decision-making process, presenting the points of contention and how to solve them.

9 Related work

Norms differ from policies in that the latter is more "high-level", that is, policies define a general attitude or approach which can be described in terms of norms. For instance, the policy "resource x is expensive and its use should be regulated" could give rise to norms being put in place, defining who is allowed or prohibited to use x. Alternatively, policies can be seen as rules (as in, for instance, [28,3]) defining when norms (that is, permissions, prohibitions and obligations) are put in place or are revoked.

Norms and policies have been used in disparate fields, ranging from security models of programming languages such as Java [37] to the management of resources in distributed systems [28]. Explicit norms and policies, as opposed to implicit ones embedded in software, define computational behaviours but allowing designers and engineers to easily change them and verify desirable properties in them [40,39]. Policies are sometimes equated with permissions (e.g., the security model of Java and the UNIX file access system), but this view is rather limited: what is not explicitly permitted is prohibited; obligations cannot be represented.

Attempts to make law systems conflict-free can be traced back to the jurisprudential practice in human society. Conflicting laws do occur and legal theorists use a diverse set of terms such as, for example, *normative inconsistencies/conflicts*, *antinomies*, *discordance*, etc., in order to describe this phenomenon. There are three classic strategies for resolving deontic conflicts by establishing a precedence relationship between norms: *lex posterior*—the most recent norm takes precedence, *lex superior*—the norm imposed by the strongest power takes

precedence, and *lex specialis*—the most specific norm takes precedence [24]. Early investigations into norm conflicts outlined in [31] describe three forms of conflict/inconsistency as *total–total*, *total–partial* and *intersection*. These are special cases of the intersection of norms as described in [21]—a permission entailing the prohibition, a prohibition entailing the permission or an overlap of both norms.

In [32,33], Sartor discusses aspects of non-monotonic reasoning in law, negation and conflict. It is pointed out that legal reasoning is often based on *prima facie* incompatible premises; this is due to the defeasibility of legal norms and the dynamics of normative systems, where new norms may contradict older ones (principle of *lex posterior*), the concurrence of multiple legal sources with normative power distributed among different bodies issuing contradicting norms (principle of *lex superior*), and semantic indeterminacy. To resolve such conflicts, it is proposed to establish an ordering among norms according to criteria such as hierarchy (*lex superior*), chronology (*lex posterior*), speciality (exception to the norm are preferred) or hermeneutics (more plausible interpretations are preferred). The work presented in [21] discusses in part these kinds of strategies, proposing conflict resolution according to the criteria mentioned above.

The work described in [7] analyses different normative conflicts—in spite of its title, the analysis is an informal one. That work differentiates between actions that are simultaneously prohibited and permitted—these are called *deontic inconsistencies*—and actions that are simultaneously prohibited and obliged—these are called *deontic conflicts*. The former is merely an "inconsistency" because a permission may not be acted upon, so no real conflict actually occurs. On the other hand, those situations when an action is simultaneously obliged and prohibited represent conflicts, as both obligations and prohibitions influence behaviours in an incompatible fashion. Our approach to detecting conflicts can capture the three forms of conflict/inconsistency of [31], viz. total-total, total-partial and intersection, respectively, when the permission entails the prohibition, when the prohibition entails the permission and when they simply overlap. Finally, we notice that the *world knowledge* explained in [7], required to relate actions, can be formally captured by our indirect norm conflicts depicted in Sect. 6.

Our approach can be contrasted with the work described in [19,20]: the norms in their policies, although in an alternative syntax, have the same components as the norms presented in this paper, and hence the same expressiveness. However, conflicts are resolved in a coarser fashion: one of the conflicting norms is "overridden", that is, it becomes void. Explicit *meta-policies* specify which of the two conflicting norms should be overridden—the curtailment policies introduced in Sect. 4.4 achieve the same effect, but constraints further refine the conditions under which the curtailment should take place. It is not clear how constraints in the norms of [19,20] affect conflict, nor how conflicts are detected—from the informal explanation given, however, only direct conflicts are addressed. Our conflict resolution is finer-grained: norms are overridden for specific values (and not completely).

There are some interesting similarities between the algorithm for norm adoption presented in this paper and those used to compute preferred extensions in abstract argumentation frameworks [6]. Norms, conflicts between norms and policies for resolving conflicts can be captured as a "normative conflict graph", as proposed in [29]. In a normative conflict graph nodes represent norms and an edge represents the notion that the norms are in conflict and that the conflict resolution policy dictates that one is preferred over the other. Such graphs are analogous to argument systems, where the nodes represent arguments and edges represent attacks between arguments. A preferred extension of an argument system is a maximal (with respect to set inclusion) admissible set of arguments, where a set of arguments is admissible if it is conflict free and each argument in the set is acceptable (i.e., for each argument that

attacks it, there is some other argument in the extension that attacks the attacker). This is a subtly stronger notion of consistency than that used in this paper; further discussion and some preliminary results in the use of Dung's semantics for normative conflict resolution can be found in [29]. It should be noted that in [29] a conflict is resolved by simply dropping norms outside of the preferred extension, but that it is possible to combine our more expressive language with this argumentation-inspired approach.

In [3], policies are used to manage the quality of service (QoS) of a network. That work formalises a simpler notion of direct normative conflicts in which constraints are not used. Interestingly, that work also presents a notion of indirect conflicts, similar to ours. Conflicts in policies are detected via abduction and an explanation is also provided as to why they are in conflict. No means to resolve conflicts is proposed in that work, though.

This paper extends and integrates previous work. A simpler version of the conflict resolution mechanism was originally proposed in [41]. However, that work did not address norms with arbitrary constraints—these were introduced in [22], together with an extension of the mechanism to resolve conflicts making use of constraints. That work also introduced policies and indirect conflicts, incorporating these concepts into a norm adoption algorithm. Finally, in [23] we introduced the norm removal mechanism.

10 Conclusions, discussion and future work

In this paper, we have proposed a representation of norms as atomic formulae whose variables may have arbitrary associated constraints. We provide mechanisms to detect and resolve conflicts between such norms, including indirect conflicts and conflicts caused by delegation of actions among agents. Norm conflicts arise when an action is simultaneously obliged and prohibited/permitted. The mechanisms are based on first-order unification and constraint satisfaction; the conflict resolution mechanism amounts to manipulating the constraints of norms to avoid overlapping values of variables—this is called the "curtailment" of variables/norms. We provided a prototypical implementation of the curtailment process as a logic program and used it to define algorithms for managing (adding and removing) norms. We made use of explicit policies to specify which of two conflicting norms should be curtailed (and under which circumstances). We provided an account of authority, and showed how these can be incorporated into our norms and associated mechanisms, allowing us to capture interesting phenomena. Lastly, we provided an example illustrating how our norm representation and associated mechanisms come together as a means to support humans making decisions in norm-regulated scenarios.

In this paper, norms only refer to atomic actions. Although this representation is useful for various realistic multi-agent scenarios, a natural extension is to allow arbitrary logical formulae to be prefixed by a deontic modality and also to allow arbitrary nesting of modalities, as in the case in full first-order deontic logics [43–45]. This extension would allow us to formalise norms such as "if one is allowed to fly then one is allowed to land and to fire missiles", as well as "one is not allowed to permit anyone to drink or smoke in the building".

In particular, we want to extend our mechanisms to cope with conditional norms [5,25]—these are of the form *Condition* $\rightarrow$ *Norms*, specifying *Condition*s (circumstances) under which certain *Norms* become active, as in, for example, $\mathsf{O}_{A:R}p(X,Y) \rightarrow \mathsf{F}_{A:R}q(X,Y)$. If this conditional norm were used with the following set of normative positions

$$\{(\mathsf{O}_{A:R}p(X,Y) \circ \{X < 20\}), (\mathsf{P}_{A:R}q(75,Y) \circ \{Y > 40\})\}$$

then a conflict would arise: the new set of normative positions obtained by applying the conditional norm would be

$$\{(\mathsf{O}_{A:R}\, p(X, Y) \circ \{X < 20\}), (\mathsf{P}_{A:R} q(75, Y) \circ \{Y > 40\}), (\mathsf{F}_{A:R} q(X, Y) \circ \{X < 20\})\}$$

There is an overlap of the values of the variables in the permission and the prohibition. Rather than applying our mechanism to resolve the conflict, we can anticipate the conflict and annotate the conditional norm with extra constraints to curtail its applicability. In our case, the conflict can be avoided if the conditional norm becomes the following set of formulae

$$\left\{ \begin{array}{l} (\mathsf{O}_{A:R}\, p(X, Y) \circ \{X \neq 75\}) \rightarrow \mathsf{F}_{A:R} q(X, Y) \\ (\mathsf{O}_{A:R}\, p(X, Y) \circ \{Y \leq 40\}) \rightarrow \mathsf{F}_{A:R} q(X, Y) \end{array} \right\}$$

We shall build on our previous work [15] in which we studied how constraints can be given different computational interpretations depending on whether they are used in the left- or right-hand side of conditional norms.

We have also noticed that our inter-role $\succ$ relationship can be generalised to capture a notion of *contextualised power/influence*. In real life situations, depending on which scenarios agents interact, their power to influence (or delegate) may be rather different. For instance, an individual may be the head of a team with a specific mission, being able to assign norms to team members, delegate tasks and be the authority to whom team members are held responsible for complying with norms. However, the team leader with a different mission may be assigned/delegated norms due, for instance, to its having a specific skill. The inter-role relationship can thus be formalised *relative* to a context: $r \succ_\kappa r'$ represents that agents adopting role r may exert influence over agents adopting role r' only in the case of norms whose actions are in κ. An example of this relative power of influence is $srg \succ_{\{fly(a)\}} sld$ (the sergeant may delegate to a soldier a norm which concerns flying aircrafts of type a) and $sld \succ_{\{fly(b)\}} srg$ (the soldier may delegate to the sergeant a norm which concerns flying aircrafts of type b, because, for instance, the soldier does not have the required training).

We would also like to exploit our mechanisms to analyse an existing set of norms with a view to detect and resolve any conflicts among its elements. In this case, the curtailment policies could be used to order the analysis process, and it would curtail first those norms which curtail the largest number of norms (that is, the norms that appear in the first position of the largest number of policies in Π). The rationale for this is that we want to preserve as much as possible of the scope of influence of each norm in the original set and if we curtail a norm which curtails various other norms, then its reduced scope of influence may not overlap with the other norms anymore. Hence, fewer conflicts will arise (as the norm which curtails other norms will have its scope reduced) and the norms which would otherwise be curtailed will be preserved in their original format.

We are exploiting our approach to support humans in mission-critical scenarios [46], including, for instance, combat and disaster recovery. We want to describe mission scripts as sets of norms which work as contracts that teams of human and software agents can peruse and reason about. Mission-critical contracts should allow for the delegation of actions and norms, via pre-established relationships between roles: we have been experimenting with special "counts as" operators [34] which neatly capture this. We also plan to explore norms to guide the development of electronic institutions [8] as well as individual software agents. This initiative would allow engineers to experiment with alternative normative specifications which would be checked for desirable properties such as conflict-freedom and feasibility (at least one norm-compliant enactment can be obtained). The specifications could then be used to synthesise norm-compliant skeletons of agents which, on their turn, could be augmented with arbitrary functionalities, along the lines of the work described in [42].

Acknowledgements This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

1. Alchourron, C. E., & Bulygin, E. (1981). The expressive conception of norms. In R. Hilpinen (Ed.), *New studies in deontic logics* (pp. 95–124). London: D. Reidel.
2. Apt, K. R. (1997). *From logic programming to prolog*. UK: Prentice-Hall.
3. Charalambides, M., Flegkas, P., Pavlou, G., Bandara, A., Lupu, E., Russo, A., Dulay, N., Sloman, M., & Rubio-Loyola, J. (2005). Policy conflict analysis for quality of service management. In *6th IEEE Workshop on Policies for District System & Networks (Policy 2005)*, June. Washington, D.C.
4. Conte, R., & Castelfranchi, C. (1995). Understanding the functions of norms in social groups through simulation. In N. Gilbert & R. Conte (Eds.), *Artificial societies: The computer simulation of social life* (pp. 252–267). London: UCL Press.
5. Dignum, F. (1999). Autonomous agents with norms. *Artificial Intelligence and Law, 7*, 69–79.
6. Dung, P. M. (1995). On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and N-person games. *Artificial Intelligence, 77*(2), 321–357.
7. Elhag, A. A. O., Breuker, J. A. P. J., & Brouwer, P. W. (2000). On the formal analysis of normative conflicts. *Information and Communication Technology Law, 9*(3), 207–217.
8. Esteva, M. (2003). *Electronic institutions: From specification to development*. PhD thesis, Universitat Politècnica de Catalunya (UPC). IIIA monography, Vol. 19.
9. Fasli, M. (2006). On the relationship between roles and power: Preliminary report. In H. Haddad (Ed.), *Proceedings of the 2006 ACM Symposium on Applied Computing (SAC)* (pp. 313–318). Dijon, France, 23–27 April 2006. ACM.
10. Fitting, M. (1990). *First-order logic and automated theorem proving*. New York, USA: Springer.
11. Ford, D. N., & Sterman, J. (1997). Expert knowledge elicitation to improve mental and formal models. Technical report WP 3953-97, Sloan School of Management, MIT.
12. French, J., & Raven, B. (1956). The bases of social power. In D. Cartwright (Ed.), *Studies in Social Power* (pp. 150–167). Michigan, USA: University of Michigan Press.
13. Gaertner, D., García-Camino, A., Noriega, P., Rodríguez-Aguilar, J.-A., & Vasconcelos, W. W. (2007). Distributed norm management in regulated multi-agent systems. In *Proceedings of the 6th International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS'07)*. Honolulu, Hawai'i, May.
14. García-Camino, A., Rodríguez-Aguilar, J.-A., Sierra, C., & Vasconcelos, W. W. (2006). A distributed architecture for norm-aware agent societies. In M. Baldoni, U. Endriss, A. Omicini, & P. Torroni (Eds.), *Proceedings of the 3rd International Worskhop on Declarative Agent Languages and Technologies (DALT 2005), Selected and Revised Papers*, *Lecture Notes in Computer Science* (Vol. 3904, pp. 89–105). Utrecht, The Netherlands: Springer, 25 July 2005, 2006.
15. García-Camino, A., Rodríguez-Aguilar, J.-A., Sierra, C., & Vasconcelos, W. W. (2006). A rule-based approach to norm-oriented programming of electronic institutions. *ACM SIGecom Exchanges, 5*(5), 33–40.
16. García-Camino, A., Rodríguez-Aguilar, J.-A., Sierra, C., & Vasconcelos, W. W. (2006). Constraint rule-based programming of norms for electronic institutions. *Journal of Autonomous Agents & Multiagent Systems* (to appear).
17. Jaffar, J., & Maher, M. J. (1994). Constraint logic programming: A survey. *Journal of Logic Programming, 19/20*, 503–581.
18. Jaffar, J., Maher, M. J., Marriott, K., & Stuckey, P. J. (1998). The semantics of constraint logic programs. *Journal of Logic Programming, 37*(1–3), 1–46.
19. Kagal, L., & Finin, T. (2005). Modeling communicative behavior using permissions and obligations. In F. Dignum, R. van Eijk, & M.-P. Huget (Eds.), *Developments in Agent Communication* (Vol. 3396, pp. 120–133). Springer.
20. Kagal, L., & Finin, T. (2007). Modeling conversation policies using permissions and obligations. *Journal of Autonomous Agents & Multiagent Systems, 14*(2), 187–206.
21. Kollingbaum, M. J., Norman, T. J., Preece, A., & Sleeman, D. H. (2006). Norm refinement: Informing the re-negotiation of contracts. In G. Boella, O. Boissier, E. Matson, & J. Vazquez-Salceda (Eds.), *ECAI*

2006 Workshop on Coordination, Organization, Institutions and Norms in Agent Systems, COIN@ECAI 2006 (pp. 46–51).
22. Kollingbaum, M. J., Vasconcelos, W. W., García-Camino, A., & Norman, T. J. (2008). Conflict resolution in norm-regulated environments via unification and constraints. In *Proceedings of the 5th International Worskhop on Declarative Agent Languages and Technologies (DALT 2007), Selected and Revised Papers*, *Lecture Notes in Computer Science* (Vol. 4897, pp. 158–174). Springer.
23. Kollingbaum, M. J., Vasconcelos, W. W., García-Camino, A., & Norman, T. J. (2008). Managing conflict resolution in norm-regulated environments. In *Proceedings of 8th Annual International Workshop "Engineering Societies in the Agents World" (ESAW 07)*, *Lecture Notes in Computer Science* (Vol. 4995, pp. 55–71). Springer.
24. Leite, J. A., Alferes, J. J., & Pereira, L. M. (2001). *Multi-dimensional dynamic knowledge representation*, *Lecture Notes in Artificial Intelligence* (Vol. 2173). Springer.
25. Makinson, D., & van der Torre, L. (2000). Input-output logics. *Journal of Philosophical Logic, 29*, 383–408.
26. Manna, Z. (1974). *Mathematical theory of computation*. Tokio, Japan: McGraw-Hill Kogakusha, Ltd.
27. McCallum, M., Vasconcelos, W. W., & Norman, T. J. (2008). Organisational change through influence. *Journal of Autonomous Agents & Multiagent Systems, 17*(2), 157–189.
28. Moffett, J., & Sloman, M. (1994). Policy conflict analysis in distributed systems management. *Journal of Organizational Computing, 4*(1), 1–22.
29. Oren, N., Luck, M., Miles, S., & Norman, T. J. (2008). An argumentation-inspired heuristic for resolving normative conflict. In *Proceedings of the 5th Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems (COIN@AAMAS)*, Estoril, Portugal.
30. Pacheco, O., & Carmo, J. (2003). A role based model for the normative specification of organized collective agency and agents interaction. *Autonomous Agents and Multi-Agent Systems, 6*(2), 145–184.
31. Ross, A. (1958). *On law and justice*. Stevens & Sons.
32. Sartor, G. (1991). The structure of norm conditions and nonmonotonic reasoning in law. In *Proceedings of the 3rd International Conference on Artificial Intelligence and Law ICAIL'91* (pp. 155–164). Oxford, U.K.
33. Sartor, G. (1993). A simple computational model for nonmonotonic and adversarial legal reasoning. In *Proceedings of the 4th International Conference on Artificial Intelligence and Law ICAIL'93* (pp. 192–201). Amsterdam, The Netherlands.
34. Searle, J. R. (1997). *The construction of social reality*. Free Press, January.
35. Sergot, M. (2001). A computational theory of normative positions. *ACM Transactions on Computational Logic, 2*(4), 581–622.
36. Shapiro, L., & Sterling, E. Y. (1994). *The art of prolog: Advanced programming techniques*. The MIT Press, April.
37. Spell, B. (2000). *Professional Java programming*. Wrox Press Inc.
38. Swedish Institute of Computer Science. (2005). *SICStus prolog*. http://www.sics.se/isl/sicstuswww/site/index.html, viewed on 10 Feb 2005 at 18.16 GMT.
39. Vasconcelos, W. W. (2004). Norm verification and analysis of electronic institutions, In J. Leite, A. Omicini, P. Torroni, P. Yolum (Eds.), *Selected and revised papers of the International Workshop on Declarative Agent Languages and Technologies (DALT), New York, U.S.A., July 2004. Lecture Notes in Computer Science*. (Vol. 3476, pp. 166–182). Springer-Verlag.
40. Vasconcelos, W. W., Esteva, M., Sierra, C., & Rodríguez-Aguilar, J. A. (2004). Verifying norm consistency in electronic institutions. In *Proceedings of the AAAI-04 Workshop on Agent Organizations: Theory and Practice* (pp. 8–14). San José, California, USA, July 25–29 2004. Technical Report WS-04-02.
41. Vasconcelos, W. W., Kollingbaum, M. J., & Norman, T. J. (2007). Resolving conflict and inconsistency in norm-regulated virtual organizations. In *Proceedings of the 6th International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS 2007)* (pp. 632–639). Hawai'i, U.S.A., May 2007. IFAAMAS.
42. Vasconcelos, W. W., Robertson, D., Sierra, C., Esteva, M., Sabater, J., & Wooldridge, M. (2004). Rapid prototyping of large multi-agent systems through logic programming. *Annals of Mathematics and Artificial Intelligence, 41*(2–4), 135–169.
43. von Wright, G. H. (1963). *Norm and action: A logical inquiry*. London: Routledge and Kegan Paul.
44. von Wright, G. H. (1967). *Logical studies*. Routledge and Kegan Paul.
45. von Wright, G. H. (1968). *An essay in deontic logic and the general theory of action*. North-Holland Publishing Company, 1968.
46. White, S. M. (2006). Requirements for distributed mission-critical decision support systems. In *Proceedings of the 13th Annual IEEE International Symposium & Workshop on Engineering of Computer-Based Systems (ECBS'06)*. Washington, D.C.
47. Wooldridge, M. (2002). *An introduction to multiagent systems*. Chichester, UK: Wiley, Feb.