Managing Conflict Resolution in Norm-Regulated Environments

M. J. Kollingbaum[1,*], W. W. Vasconcelos[1,†], A. García-Camino[2,◇], and T. J. Norman[1,‡]

[1]Dept. of Computing Science, Univ. of Aberdeen, Aberdeen AB24 3UE, UK
{*mkolling,† wvasconc,‡ tnorman}@csd.abdn.ac.uk
[2]IIIA-CSIC, Campus UAB 08193 Bellaterra, Spain
◇andres@iiia.csic.es

Abstract. Norms, that is, obligations, permissions and prohibitions, are a useful abstraction to specify and regulate the actions of self-interested software agents in open, heterogeneous systems. Any realistic account of norms must address their dynamic nature: the norms associated with agents will change as agents act (and interact) – prohibitions can be lifted, obligations can be fulfilled and permissions can be revoked as a result of agents' behaviours. These norms may at times *conflict* with one another, that is, an action may be simultaneously prohibited and obliged (or prohibited and permitted). Norm conflicts prevent agents from rationally deciding on their behaviour. In this paper, we present mechanisms to detect and resolve normative conflicts. We achieve more expressiveness, precision and realism in our norms by using *constraints* over first-order variables. The mechanism to detect and resolve norm conflicts takes into account such constraints and is based on first-order unification and constraint satisfaction. We also explain how the mechanisms can be deployed in the management of an explicit account of all norms associated with a society of agents.

1 Introduction

Norms, that is, obligations, permissions and prohibitions, are a useful abstraction to specify and regulate the observable behaviour in electronic environments of self-interested, heterogeneous software agents. As expressed in [2, 20], these agents are usually designed by various parties who may not entirely trust each other. Norms also support the establishment of organisational structures for coordinated resource sharing and problem solving [6, 15]. Such virtual organisations may be supplemented with an explicit and separate set of norms. These norms regulate the behaviour of agents and impose restrictions and preferences on the choices from their behavioural repertoire.

Norm-regulated environments may experience problems when norms assigned to their agents are in *conflict* – actions that are forbidden, may, at the same time, be also obliged and/or permitted. For example, "Agent X is permitted to $send\_bid(ag_1, 20)$" and "Agent ag_2 is prohibited from doing $send\_bid(Y, Z)$"

(where X, Y and Z are variables and ag_1, ag_2 and 20 are constants) are two norms in conflict regarding action *send_bid*/2. Because normative conflicts may "paralyse" a software agent, seriously compromising its autonomy, we must provide means to automatically detect and resolve such conflicts.

In this paper, we propose a means to automatically detect and resolve norm conflicts. We make use of first-order unification [5] to find out if and how norms overlap in their *scope of influence* [12]. If such a conflict is detected, a resolution can be found by proposing a *curtailment* of the conflicting norms. We curtail norms by adding constraints. For example, if we add constraint $X \neq ag_2$ to the permission mentioned above, we curtail this norm such that it is relevant to any agent other than agent ag_2. The scope of influence of the permission becomes restricted and does not overlap with the influence of the prohibition.

In the following sections we provide an overview of a model of norm-governed agency and formally define concepts such as actions, norms and the global normative state of a society. Based on this model, we explain the detection and resolution of conflicts between norms, and provide algorithms for managing normative states of a society.

2 Norm-Governed Agency

Our model of norm-governed agency assumes that agents take on roles within a society or organisation and that these roles are defined by a set of norms. Roles, as used in, *e.g.*, [16], help us abstract from individual agents, defining a pattern of behaviour to which any agent that adopts a role ought to conform. 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. Central to our model is the concept of actions performed by agents:

Definition 1. *$\langle a : r, \bar{\varphi}, t \rangle$ represents a specific action $\bar{\varphi}$ (a ground first-order atomic formula), performed by $a \in Agents$ adopting $r \in Roles$ at time $t \in I\!N$.*

Although agents are regarded as performing their actions in a distributed fashion (and, therefore, contributing to the enactment of the overall system), we propose a global account for all the actions taking place. It is, therefore, important to record the authorship of actions and the time when it occurs. The set Ξ stores such tuples recording actions of agents and 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 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 distributedly.

2.1 Norm Specification

We extend the notion of a norm as presented in [22]. We adopt the notation of [16] for specifying norms, complementing it with *constraints* [11]. Constraints are used to *refine* the influence of norms on specific actions. We shall make use of numbers and arithmetic functions to build terms. Arithmetic functions may appear infix, following their usual conventions[1]. A syntax for constraints is introduced as follows:

Definition 3. *Constraints, represented as* γ*, are any construct of the form* $\tau \lhd \tau'$*, where* $\lhd \in \{=, \neq, >, \geq, <, \leq\}$.

Norms are defined as follows:

Definition 4. *A norm* ω *is a tuple* $\langle \nu, t_d, t_a, t_e \rangle$*, where* ν *is any construct of the form* $\mathsf{O}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i$ *(an obligation),* $\mathsf{P}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i$ *(a permission) or* $\mathsf{F}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i$ *(a prohibition), where* τ_1, τ_2 *are terms,* φ *is a first-order atomic formula and* γ_i*,* $0 \leq i \leq n$*, are constraints. The elements* $t_d, t_a, t_e \in I\!N$ *are, respectively, the time when* ν *was declared (introduced), when* ν *becomes active and when* ν *expires,* $t_d \leq t_a \leq t_e$.

Term τ_1 identifies the agent(s) to whom the norm is applicable and τ_2 is the role of such agent(s). $\mathsf{O}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i$ thus represents an obligation on agent τ_1 taking up role τ_2 to bring about φ, subject to constraints γ_i, $0 \leq i \leq n$. The γ_i's express constraints on those variables occurring in φ. In the definition above we only cater for conjunctions of constraints. If disjunctions are required then a norm must be established for each disjunct. For instance, if we required the norm $\mathsf{P}_{A:R}move(A) \wedge A < 10 \vee A = 15$ then we must break it into two norms $\mathsf{P}_{A:R}move(A) \wedge A < 10$ and $\mathsf{P}_{A:R}move(A) \wedge A = 15$. We assume an implicit universal quantification over variables in ν. For instance, $\mathsf{P}_{A:R}p(X, b, c)$ stands for $\forall A \in Agents.\forall R \in Roles.\forall X.\mathsf{P}_{A:R}p(X, b, c)$.

We propose to formally represent the normative positions of all agents, taking part in a virtual society, from a global perspective. By "normative position" we mean the "social burden" associated with individuals [8], that is, their obligations, permissions and prohibitions:

Definition 5. *A global normative state* Ω *is a finite and possibly empty set of tuples* $\omega = \langle \nu, t_d, t_a, t_e \rangle$.

A global normative state, expressed by Ω, complements the enactment state of a virtual society, expressed by Ξ, 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 simple but useful approach is reported in [7]: production rules generically depict how norms should be updated to reflect what agents have done and which norms currently hold. Similarly to Ξ, we use a single normative state Ω to simplify our exposition; however, we can also slice Ω into various sub-sets and manage them distributedly.

[1] We adopt Prolog's convention [1] using strings starting with a capital letter to represent variables and strings starting with a small letter to represent constants.

3 Norm Conflicts

We provide definitions for conflicts, their detection and resolution using constraints. Constraints confer more expressiveness and precision on norms, but the mechanisms for detection and resolution must factor them in. We use first-order unification [5] and constraint satisfaction [11] 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 [5] . In addition, we describe, how substitutions are applied to norm such as obligations, permissions and prohibitions. (X stands for either 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. $(\mathsf{X}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i) \cdot \sigma = (\mathsf{X}_{(\tau_1\cdot\sigma):(\tau_2\cdot\sigma)}\varphi \cdot \sigma) \wedge \bigwedge_{i=0}^{n}(\gamma_i \cdot \sigma)$.
5. $\langle \nu, t_d, t_a, t_e \rangle \cdot \sigma = \langle (\nu \cdot \sigma), t_d, t_a, t_e \rangle$

A substitution σ is a *unifier* of two terms τ_1, τ_2, if $\tau_1 \cdot \sigma = \tau.\sigma$. Unification is a fundamental problem in automated theorem proving and many algorithms have been proposed [5], recent work proposing means to obtain unifiers efficiently. We shall 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 $unify(\tau_i, \tau'_i, \sigma), 0 \le i \le n$.*

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.

3.1 Conflict Detection

Unification allows us to detect a conflict between norms. Norm conflict is formally defined as follows:

Definition 8. *A conflict arises between $\omega, \omega' \in \Omega$ under a substitution σ, denoted as* **conflict**$(\omega, \omega', \sigma)$*, iff the following conditions hold:*

1. $\omega = \langle (\mathsf{F}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i), t_d, t_a, t_e \rangle$, $\omega' = \langle (\mathsf{O}_{\tau'_1:\tau'_2}\varphi' \wedge \bigwedge_{i=0}^{n} \gamma'_i), t'_d, t'_a, t'_e \rangle$,
2. $unify(\langle \tau_1, \tau_2, \varphi \rangle, \langle \tau'_1, \tau'_2, \varphi' \rangle, \sigma)$, $satisfy(\bigwedge_{i=0}^{n} \gamma_i \wedge (\bigwedge_{i=0}^{m} \gamma'_i \cdot \sigma))$
3. $overlap(t_a, t_e, t'_a, t'_e)$.

That is, a conflict occurs if *i)* a substitution σ can be found that unifies the variables of two norms[2], and *ii)* the conjunction $\bigwedge_{i=0}^{n} \gamma_i \wedge (\bigwedge_{i=0}^{m} \gamma'_i) \cdot \sigma)$ of constraints from both norms can be satisfied[3] (taking σ under consideration), and *iii)* the activation period of the norms overlap. The *overlap* relationship holds if *i)* $t_a \leq t'_a \leq t_e$; or *ii)* $t'_a \leq t_a \leq t'_e$.

For instance, $\mathsf{P}_{A:R} p(c, X) \wedge X > 50$ and $\mathsf{F}_{a:b} p(Y, Z) \wedge Z < 100$ are in conflict. We can obtain a substitution $\sigma = \{A/a, R/b, Y/c, X/Z\}$ which shows how they overlap. Being able to construct such a unifier 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 permission has a constraint $X > 50$ and the prohibition has $Z < 100$. By using the substitution X/Z, we see that $50 < X < 100$ and $50 < Z < 100$ represent ranges of values for variables X and Z where a conflict will occur.

For convenience (and without any loss of generality) we assume that our norms are in a special format: any non-variable term τ occurring in ω is replaced by a fresh variable X (not occurring anywhere in ω) and a constraint $X = \tau$ is added to ω. This transformation can be easily automated by scanning ω from left to right, collecting all non-variable terms $\{\tau_1, \ldots, \tau_n\}$; then we add $\wedge_{i=1}^{n} X_i = \tau_i$ to ν. For example, norm $\mathsf{P}_{A:R} p(c, X) \wedge X > 50$ is transformed into $\mathsf{P}_{A:R} p(C, X) \wedge X > 50 \wedge C = c$.

3.2 Conflict Resolution

In order to resolve a conflict, a machinery has to be put in place that transforms the set of norms Ω into a conflict-free set Ω'. We curtail norms by manipulating constraints. In figure 1, we show how a curtailment of the prohibition changes its scope of influence and thus eliminates the overlap between the two norms. Specific constraints are added to the prohibition in order to perform this curtailment. These additional constraints are derived from the permission. The scope of the permission is determined by the constraints $X = a$ and $Y = r$, restricting the set of bindings for variables X and Y to values a and r. Adding a constraint $Y \neq r$ to the prohibition curtails its scope of influence and eliminates the overlap with the scope of influence of the permission.

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 (possibly empty) set of curtailed norms:

---

[2] A similar definition is required to address the case of conflict between a prohibition and a permission – the first condition should be changed to $\omega' = \langle (\mathsf{P}_{\tau'_1:\tau'_2} \varphi' \wedge \bigwedge_{i=0}^{n} \gamma'_i), t'_d, t'_a, t'_e \rangle$. The rest of the definition remains the same.

[3] We assume an implementation of the *satisfy* relationship based on "off-the-shelf" constraint satisfaction libraries such as those provided by SICStus Prolog [21] and it holds if the conjunction of constraints is satisfiable.

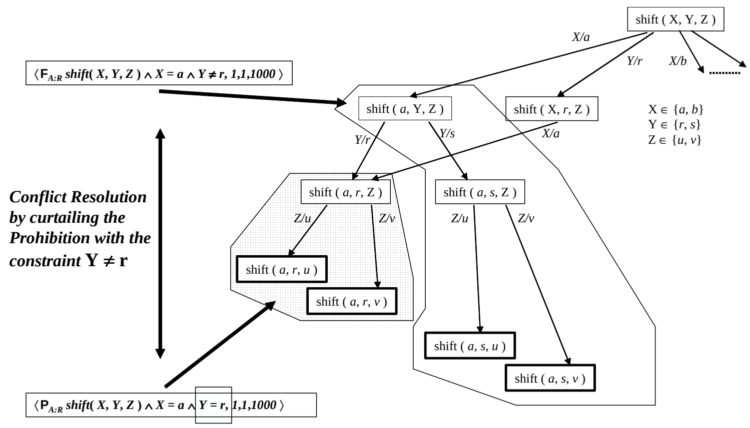

Fig. 1. Conflict Resolution with Curtailment

Definition 9. *Relationship* **curtail**$(\omega, \omega', \Omega)$*, where* $\omega = \langle \mathsf{X}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i, t_d, t_a, t_e\rangle$ *and* $\omega' = \langle \mathsf{X}'_{\tau'_1:\tau'_2}\varphi' \wedge \bigwedge_{j=0}^{m} \gamma'_j, t'_d, t'_a, t'_e\rangle$ *(*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:*

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* $\Omega = \{\omega^c_0, \ldots, \omega^c_m\}$*, where* $\omega^c_j = \langle \mathsf{X}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i \wedge (\neg\gamma'_j \cdot \sigma), t_d, t_a, t_e\rangle$*,* $0 \leq j \leq m$*.*

In order to curtail ω, thus avoiding any overlapping of 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.

By combining the constraints of $\nu = \mathsf{X}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i$ and $\nu' = \mathsf{X}'_{\tau'_1:\tau'_2}\varphi' \wedge \bigwedge_{j=0}^{m} \gamma'_j$, we obtain the curtailed norm $\nu^c = \mathsf{X}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i \wedge \neg(\bigwedge_{j=0}^{m} \gamma'_j \cdot \sigma)$. The following equivalences hold:

$$\mathsf{X}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i \wedge \neg(\bigwedge_{j=0}^{m} \gamma'_j \cdot \sigma) \equiv \mathsf{X}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i \wedge (\bigvee_{j=0}^{m} \neg\gamma'_j \cdot \sigma)$$

That is, $\bigvee_{j=0}^{m}(\mathsf{X}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i \wedge \neg(\gamma'_j \cdot \sigma))$. This shows that each constraint of ν' leads to a possible solution for the resolution of a conflict and a possible curtailment of ν. The curtailment thus produces a set of curtailed norms $\nu^c_j = \mathsf{X}_{\tau_1:\tau_2}p(t_1, \ldots, t_n) \wedge \bigwedge_{i=0}^{n} \gamma_i \wedge \neg\gamma'_j \cdot \sigma, 0 \leq j \leq m$.

Although each of the ν^c_j, $0 \leq j \leq m$, represents a solution to the norm conflict, we advocate that *all* of them have to be added to Ω in order to replace the curtailed norm. This would allow a preservation of as much of the original scope of the curtailed norm as possible. As an illustrative example, let us suppose an Ω with the following norm ω,

$$\Omega = \{\langle \mathsf{F}_{A:R}p(C, X) \wedge C = c \wedge X > 50, t_d, t_a, t_e\rangle\}$$

If we try to introduce a new norm $\omega' = \langle \mathsf{P}_{B:S} p(Y, Z) \wedge B = a \wedge S = r \wedge Z > 100, t'_d, t'_a, t'_e \rangle$ to Ω, then we have a conflict. This conflict can be resolved by curtailing one of the two conflicting norms. The constraints in ω' are used to create such a curtailment. Using σ, we construct copies of ω, but adding $\neg\gamma'_i \cdot \sigma$ to them. In our example, the constraint $Z > 100$ becomes $\neg(Z > 100) \cdot \sigma$, that is, $X \leq 100$. With the three constraints contained in ω', three options for curtailing ω can be constructed. A new Ω' is constructed, containing *all* the options for curtailment:

$$\Omega' = \left\{ \begin{array}{l} \langle \mathsf{P}_{B:S} p(Y, Z) \wedge B = a \wedge S = r \wedge Z > 100, t'_d, t'_a, t'_e \rangle \\ \langle \mathsf{F}_{A:R} p(C, X) \wedge C = c \wedge X > 50 \ \wedge A \neq a, t_d, t_a, t_e \rangle \\ \langle \mathsf{F}_{A:R} p(C, X) \wedge C = c \wedge X > 50 \ \wedge R \neq r, t_d, t_a, t_e \rangle \\ \langle \mathsf{F}_{A:R} p(C, X) \wedge C = c \wedge X > 50 \ \wedge X \leq 100, t_d, t_a, t_e \rangle \end{array} \right\}$$

For each $\neg\gamma'_i \cdot \sigma$ ($A \neq a$, $R \neq r$ and $X \leq 100$ in our example), the original prohibition is extended with one of these constraints and added as a new, more restricted prohibition to Ω'. Each of these options represents a part of the scope of influence regarding actions of the original prohibition ω, restricted in a way such that a conflict with the permission is avoided. In order to allow a check whether any other action that was prohibited by ω is prohibited or not, it is necessary to make all three prohibitions available in Ω'. If there are other conflicts, additional curtailments may be necessary.

3.3 An Implementation of Norm Curtailment

We show in Figure 2 a prototypical implementation of the curtailment process 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

1 $\mathbf{curtail}(\omega, \omega', \Omega) \leftarrow$
2 $\quad \omega = \langle \mathsf{X}_{\tau_1:\tau_2} \varphi \wedge \bigwedge_{i=0}^{n} \gamma_i, t_d, t_a, t_e \rangle \wedge$
3 $\quad \omega' = \langle \mathsf{X}'_{\tau'_1:\tau'_2} \varphi' \wedge \bigwedge_{j=0}^{m} \gamma'_j, t'_d, t'_a, t'_e \rangle \wedge$
4 $\quad \mathbf{conflict}(\omega, \omega', \sigma) \wedge$
5 $\quad \mathit{merge}([(\neg\gamma'_0 \cdot \sigma), \ldots, (\neg\gamma'_m \cdot \sigma)], (\bigwedge_{i=0}^{n} \gamma_i), \widehat{\Gamma}) \wedge$
6 $\quad \mathit{setof}(\langle \mathsf{X}_{\tau_1:\tau_2} \varphi \wedge \Gamma, t_d, t_a, t_e \rangle, \mathit{member}(\Gamma, \widehat{\Gamma}), \Omega)$
7 $\mathbf{curtail}(\omega, \omega', \{\omega\})$

8 $\mathit{merge}([], \_, [])$
9 $\mathit{merge}([(\neg\gamma' \cdot \sigma) | Gs], (\bigwedge_{i=0}^{n} \gamma_i), [\Gamma | \widehat{\Gamma}]) \leftarrow$
10 $\quad \mathit{satisfy}((\bigwedge_{i=0}^{n} \gamma_i) \wedge (\neg\gamma' \cdot \sigma)) \wedge$
11 $\quad \Gamma = (\bigwedge_{i=0}^{n} \gamma_i) \wedge (\neg\gamma' \cdot \sigma) \wedge$
12 $\quad \mathit{merge}(Gs, (\bigwedge_{i=0}^{n} \gamma_i), \widehat{\Gamma})$
13 $\mathit{merge}([\_ | Gs], (\bigwedge_{i=0}^{n} \gamma_i), \widehat{\Gamma}) \leftarrow$
14 $\quad \mathit{merge}(Gs, (\bigwedge_{i=0}^{n} \gamma_i), \widehat{\Gamma})$

Fig. 2. Implementation of **curtail** as a Logic Program

an auxiliary predicate *merge*/3. 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*/3 which, as the name suggests, merges the conjunction of γ_i's with the negated constraints γ'_j's. Line 6 assembles Ω by collecting the members Γ of the list $\widehat{\Gamma}$ and using them to create curtailed versions of ω. The elements of the list $\widehat{\Gamma}$ assembled via *merge*/3 are of the form $(\bigwedge_{i=0}^{n} \gamma_i) \wedge (\neg\gamma'_j \cdot \sigma)$ – additionally, in our implementation we check if each element is satisfiable[4] (line 10). The rationale for this is that there is no point in creating a norm which will never be applicable as its constraints cannot be satisfied, so these are discarded during their preparation.

3.4 Curtailment Policies

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

Definition 10. *A policy* π *is a tuple* $\langle \omega, \omega', (\bigwedge_{i=0}^{n} \gamma_i) \rangle$ *establishing that* ω *should be curtailed (and* ω' *should be preserved), if* $(\bigwedge_{i=0}^{n} \gamma_i)$ *hold.*

A sample policy is $\langle \langle \mathsf{F}_{A:R} p(X,Y), T_d, T_a, T_e \rangle, \langle \mathsf{P}_{A:R} p(X,Y), T'_d, T'_a, T'_e \rangle, (T_d < T'_d) \rangle$. It expresses that any prohibition held by any agent that corresponds to the pattern $\mathsf{F}_{A:R} p(X,Y)$ has to be curtailed, if the additional constraint, which expresses that the prohibition's time of declaration T_d precedes that of the permission's T'_d, holds. Adding constraints to policies allows us a fine-grained control of conflict resolution, capturing classic forms of deontic conflict resolution – the constraint in the example establishes a precedence relationship between the two norms known as *legis posterior* (see section 6 for more details). We shall represent a set of such policies as Π.

3.5 Management of Normative States

The algorithm shown in figure 3 depicts how to obtain a conflict-free set of norms. It describes how an originally conflict-free (possibly empty) set Ω can be extended in a fashion that resolves any emerging conflicts during norm adoption. With that, a conflict-free Ω is always transformed into a conflict-free Ω' that may contain curtailments. The algorithm makes use of a set Π of policies determining how the curtailment of conflicting norms should be done. Policies determine whether the new norm ω is curtailed in case of a conflict or whether a curtailment of one of the existing $\omega' \in \Omega$ should be done. When a norm is curtailed, a set of new norms replace the original norm. This set of norms is collected into Ω'' by **curtail**$(\omega, \omega', \Omega'')$. A curtailment takes place if there is a conflict between ω and ω'. This test creates a unifier σ that is re-used in the policy test. When

[4] In our implementation we made use of SICStus Prolog [21] constraint satisfaction libraries [10].

[5] In [22], for instance, prohibitions are always curtailed. This ensures the choices on the agents' behaviour are kept as open as possible.

```
algorithm adoptNorm(ω, Ω, Π, Ω')
input ω, Ω, Π
output Ω'
begin
  Ω' := ∅
  if Ω = ∅ then Ω' := Ω ∪ {ω}
  else
    for each ω' ∈ Ω do
      // test for conflict
      if unify(ω, ω', σ) then
        // test policy
        if ⟨ω_π, ω'_π, (⋀_{i=0}^{n} γ_i)⟩ ∈ Π and
          unify(ω, ω_π, σ) and
          unify(ω', ω'_π, σ) and
          satisfy(⋀_{i=0}^{n}(γ_i · σ))
        then
          curtail(ω, ω', Ω'')
          Ω' := Ω ∪ Ω''
        else
          // test policy
          if ⟨ω'_π, ω_π, (⋀_{i=0}^{n} γ_i)⟩ ∈ Π and
            unify(ω, ω_π, σ) and
            unify(ω', ω'_π, σ) and
            satisfy(⋀_{i=0}^{n}(γ_i · σ))
          then
            curtail(ω', ω, Ω'')
            Ω' := (Ω − {ω'}) ∪ ({ω} ∪ Ω'')
    endfor
end
```

Fig. 3. Norm Adoption Algorithm

checking for a policy that is applicable, the algorithm uses unification to check *(a)* whether ω matches/unifies with ω_π and ω' with ω'_π; and *(b)* whether the policy constraints hold under the given σ. If a previously agreed policy in Π determines that the newly adopted norm ω is to be curtailed in case of a conflict with an existing $\omega' \in \Omega$, then the new set Ω' is created by adding Ω'' (the curtailed norms) to Ω. If the policy determines a curtailment of an existing $\omega' \in \Omega$ when a conflict arises with the new norm ω, then a new set Ω' is formed by *a)* removing ω' from Ω and *b)* adding ω and the set Ω''.

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 *roll back* the normative state to a previous form. 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 11. *$\mathcal{H}$ is a non-empty and finite sequence of tuples $\langle i, \Omega, \omega, \pi \rangle$, where $i \in I\!N$ represents the order of the tuples, Ω is a normative state, ω is a norm and π is a policy.*

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. $\mathcal{H} = \langle 0, \emptyset, \omega, \pi \rangle \bullet \mathcal{H}'$
2. $\mathcal{H} = \mathcal{H}' \bullet \langle i, \Omega', \omega', \pi' \rangle \bullet \langle i+1, \Omega'', \omega'', \pi'' \rangle \bullet \mathcal{H}''$
3. $adoptNorm(\omega_i, \Omega_i, \{\pi_i\}, \Omega_{i+1})$

The first condition establishes the first element of a history to be an empty Ω. The second condition establishes that the tuples are completely ordered on their first component. The third 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 adopting policy π_i.

$\mathcal{H}$ is required to allow the retraction of a norm in an ordered fashion, as not only the norm itself has to be removed but also all the curtailments it caused when it was introduced in Ω. $\mathcal{H}$ contains a tuple $\langle i, \Omega, \omega, \pi \rangle$ that indicates the introduction of norm ω and, therefore, provides us with a normative state Ω *before* the introduction of ω. The effect of the introduction of 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 Figure 4 as algorithm *removeNorm* describing how to remove a norm ω given a history $\mathcal{H}$; the algorithm 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,

```
algorithm removeNorm(ω, H, Ω, H')
input ω, H
output Ω, H'
begin
  if H = H' • ⟨k, Ω_k, ω, π_k⟩ • ··· • ⟨n, Ω_n, ω_n, π_n⟩
  then
    begin
      Ω := Ω_k
      for i = k + 1 to n do
        adoptNorm(ω_i, Ω, {π_i}, Ω')
        H' := H' • ⟨i, Ω, ω_i, π_i⟩
        Ω := Ω'
      endfor
    end
  else
    begin
      H = H'' • ⟨n, Ω_n, ω_n, π_n⟩
      Ω := Ω_n, H' := H
    end
end
```

Fig. 4. Algorithm to Remove Norms

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 $\mathcal{H}$.

4 Norm-Aware Agent Societies

With a set Ω that reflects a conflict-free global normative situation, agents can test whether their actions are norm-compliant. In order to check actions for norm-compliance, we, again, use unification. If an action unifies with a norm, then it is within its scope of influence:

Definition 12. $\langle a : r, \bar{\varphi}, t\rangle$, *is within the scope of influence of* $\langle \mathsf{X}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n}\gamma_i,$ $t_d, t_a, t_e\rangle$ *(where* X *is either* O*,* P *or* F*) iff the following conditions hold:*

1. *unify*$(\langle a, r, \bar{\varphi}\rangle, \langle \tau_1, \tau_2, \varphi\rangle, \sigma)$ *and satisfy*$(\bigwedge_{i=0}^{n}\gamma_i \cdot \sigma)$
2. $t_a \leq t \leq t_e$

This definition can be used to establish a predicate `check`/2, which holds if its first argument, a candidate action (in the format of the elements of Ξ of Def. 2), is within the influence of an prohibition ω, its second parameter. Figure 5 shows

$$
\begin{array}{l}
\texttt{check}(Action, \omega) \leftarrow \\
\quad Action = \langle a{:}r, \bar{\varphi}, t\rangle \wedge \\
\quad \omega = \langle(\mathsf{F}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n}\gamma_i), t_d, t_a, t_e\rangle \wedge \\
\quad unify(\langle a, r, \bar{\varphi}\rangle, \langle \tau_1, \tau_2, \varphi\rangle, \sigma) \wedge satisfy(\bigwedge_{i=0}^{n}\gamma_i \cdot \sigma) \wedge \\
\quad t_a \leq t \leq t_e
\end{array}
$$

Fig. 5. Check if Action is within Influence of a Prohibition

the definition of this relationship as a logic program. Similarly to the check of conflicts between norms, it tests *i)* if the agent performing the action and its role unify with the appropriate terms τ_1, τ_2 of ω; *ii)* if the actions $\bar{\varphi}, \varphi$ themselves unify; and *iii)* the conjunction of the constraints of both norms can be satisfied, all under the same unifier σ. Lastly, it checks if the time of the action is within the norm temporal influence.

5 Indirect Conflicts

In our previous discussion, norm conflicts were detected via a direct comparison of atomic formulae representing actions. However, conflicts and inconsistencies may also arise *indirectly* via relationships among actions. For instance, if we consider that an agent holds the two norms $\mathsf{P}_{A:R}p(X)$ and $\mathsf{F}_{A:R}q(X,X)$ and that the action $p(X)$ amounts to the action $q(X,X)$, then we can rewrite the permission as $\mathsf{P}_{A:R}q(X,X)$ and identify an *indirect* conflict between these two norms. We use a set of *domain axioms* in order to declare such domain-specific relationships between actions:

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

In order to accommodate indirect conflicts between norms based on domain-specific relationships of actions, we have to adapt our conflict detection mechanism. With the introduction of domain axioms, a test has to be performed for conflict involving each of the conjuncts in the right-hand side of the formula representing the domain axiom. For example, if the set of domain axioms is $\Delta = \{(p(X) \to q(X,X) \wedge r(X,Y))\}$ and we have a permission $\langle \mathsf{P}_{A:R}p(X), t_d, t_a, t_e\rangle$, then, according to our domain axiom, actions $q(X,X)$ and $r(X,Y)$ are also permitted. If there are two prohibitions $\langle \mathsf{F}_{A:R}q(X,X), t_d, t_a, t_e\rangle$ and $\langle \mathsf{F}_{A:R}r(X,Y), t_d, t_a, t_e\rangle$, then an indirect conflict occurs between the permission and both prohibitions.

We now revisit Definition 8 and extend it to account for relationships among actions:

Definition 14. *An indirect conflict arises between two norms ω, ω' under a set of domain axioms Δ, denoted as* **conflict**$^*(\Delta, \omega, \omega')$*, iff:*

1. **conflict**$(\omega, \omega', \sigma)$*, or*
2. $\omega = \langle (\mathsf{X}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i), t_d, t_a, t_e\rangle$*, there is an axiom* $(\varphi' \to (\varphi'_1 \wedge \cdots \wedge \varphi'_m)) \in \Delta$ *such that unify*$(\varphi, \varphi', \sigma')$*, and* $\bigvee_{i=1}^{m}$ **conflict**$^*(\Delta, \langle (\mathsf{X}_{\tau_1:\tau_2}\varphi'_i \wedge \bigwedge_{i=0}^{n} \gamma_i), t_d, t_a, t_e\rangle \cdot \sigma', \omega')$*,*

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 $(\varphi \to (\varphi'_1 \wedge \cdots \wedge \varphi'_m)) \in \Delta$, then we "transfer" the norm from φ to $\varphi'_1, \ldots, \varphi'_m$, thus obtaining $\langle (\mathsf{X}_{\tau_1:\tau_2}\varphi'_i \wedge \bigwedge_{i=0}^{n} \gamma_i), t_d, t_a, t_e\rangle, 1 \leq i \leq m$. If we (recursively) find an indirect conflict between ω' and at least one of these norms (that is, a disjunction), then an indirect conflict arises between the original norms ω, 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'_is in the recursive call(s).

Domain axioms may also accommodate the delegation of actions between agents. Such a delegation transfers norms across the agent community and, with that, also conflicts. We introduce a special logical operator $\varphi \xrightarrow{\tau_1:\tau_2\ \tau'_1:\tau'_2} (\varphi'_1 \wedge \cdots \wedge \varphi'_n)$ to represent that agent τ_1 adopting role τ_2 can transfer any norms on action φ to agent τ'_1 adopting role τ'_2, which should carry out actions $\varphi'_1 \wedge \cdots \wedge \varphi'_n$ instead. We formally capture the meaning of this operator as follows:

3. $\omega = \langle (\mathsf{X}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i), t_d, t_a, t_e \rangle$*, there is a delegation axiom* $(\varphi \xrightarrow{\tau_1:\tau_2\ \tau_1':\tau_2'} (\varphi_1' \wedge \cdots \wedge \varphi_m')) \in \Delta$*, s.t. unify*$(\langle \varphi, \tau_1, \tau_2 \rangle, \langle \varphi', \tau_1', \tau_2' \rangle, \sigma')$*, and* $\bigvee_{i=1}^{m} \textbf{conflict}^*(\Delta, \langle (\mathsf{X}_{\tau_1':\tau_2'}\varphi_i' \wedge \bigwedge_{i=0}^{n} \gamma_i), t_d, t_a, t_e \rangle \cdot \sigma', \omega')$

That is, we obtain a domain 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 $\tau_1':\tau_2'$. The new norms are recursively checked and if at least one of them conflicts with ω', then an indirect conflict arises. A mechanism should be added to detect loops in delegation.

6 Related Work

Efforts to keep law systems conflict-free can be traced back to the jurisprudential practice in human society. Inconsistency in law is an important issue 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: *legis posterior* – the most recent norm takes precedence, *legis superior* – the norm imposed by the strongest power takes precedence, and *legis specialis* – the most specific norm takes precedence [14]. Early investigations into norm conflicts were outlined in [17], describing three forms of conflict/inconsistency as *total-total*, *total-partial* and *intersection*. These are special cases of the intersection of norms as described in [13] – a permission entailing the prohibition, a prohibition entailing the permission or an overlap of both norms.

In [18, 19], aspects of legal reasoning such as nonmonotonic reasoning in law, negation and conflict are discussed. It is pointed out that legal reasoning is often based on *prima facie* incompatible premises, which is due to the defeasibility of legal norms and the dynamics of normative systems, where new norms may contradict older ones (principle of legis posterior), the concurrence of multiple legal sources with normative power distributed among different bodies issuing contradicting norms (principle of legis superior), and semantic indeterminacy. To resolve such conflicts, it is proposed to establish an ordering among norms according to criteria such as hierarchy (legis superiori), chronology (legis posteriori), speciality (exception to the norm are preferred) or hermeneutics (more plausible interpretations are preferred).

The work presented in [13] discusses in part these kinds of strategies, proposing conflict resolution according to the criteria mentioned above.

In [4] we find an analysis of different normative conflicts (albeit an informal one, in spite of its title) in which the authors suggest that a deontic inconsistency arises when an action is simultaneously permitted and prohibited – since a permission may not be acted upon, no real conflict actually occurs. The situations when an action is simultaneously obliged and prohibited are, however, deontic conflicts, as both obligations and prohibitions influence behaviours in a conflicting fashion. We notice that our approach to detecting deontic conflict can capture the three forms of conflict/inconsistency of [17], *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 [4], required to relate actions, can be formally captured by our indirect norm conflicts depicted in Section 5.

The work presented in this paper is an adaptation and extension of [22, 13] and [9], also providing an investigation into deontic modalities for representing normative concepts [3, 20]. In [22], a conflict detection and resolution based on unification is introduced: we build on that research, introducing constraints to the mechanisms proposed in that work.

7 Conclusions and Future Work

We have presented a novel mechanism to detect and resolve conflicts in norm-regulated environment. Such conflicts arise when an action is simultaneously obliged and prohibited or, alternatively, when an action is permitted and prohibited. We introduce norms as first-order atomic formulae to whose variables we can associate arbitrary constraints – this allows for more expressive norms, with a finer granularity and greater precision. The proposed mechanism is based on first-order unification and constraint satisfaction algorithms, extending the work of [22], addressing a more expressive class of norms. Our conflict resolution mechanism amounts to manipulating the constraints of norms to avoid overlapping values of variables – this is called the "curtailment" of variables/norms. A prototypical implementation of the curtailment process is given as a logic program and is used in the management of the normative state of an agent society. We have also introduced an algorithm to manage the adoption of possibly conflicting norms, whereby explicit policies depict how the curtailment between specific norms should take place, as well as an algorithm depicting how norms should be removed, thus un-doing the effects of past curtailments.

We are currently exploiting our approach in mission-critical scenarios [23], including, for instance, combat and disaster recovery (*e.g.* extreme weather conditions and urban terrorism). Our goal is to describe mission scripts as sets of norms: these will work as contracts that teams of (human and software) agents can peruse and make sense of. Mission-critical contracts should allow for the delegation of actions and norms, via pre-established relationships between roles: we have been experimenting with special "count as" operators which neatly capture this. Additionally, our mission-critical contracts should allow the representation of plan scripts with the breakdown of composite actions down to the simplest atomic actions. Norms associated with composite actions will be distributed across the composite actions, possibly being delegated to different agents and/or roles.

Acknowledgements: This research is continuing through participation in the International Technology Alliance sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence (`http://www.usukita.org`). García-Camino enjoys an I3P grant from the Spanish Council for Scientific Research (CSIC).

References

1. K. R. Apt. *From Logic Programming to Prolog.* Prentice-Hall, U.K., 1997.
2. A. Artikis, L. Kamara, J. Pitt, and M. Sergot. A Protocol for Resource Sharing in Norm-Governed Ad Hoc Networks. volume 3476 of *LNCS*. Springer-Verlag, 2005.
3. F. Dignum. Autonomous Agents with Norms. *Artificial Intelligence and Law*, 7:69–79, 1999.
4. A.A.O. Elhag, J.A.P.J. Breuker, and P.W. Brouwer. On the Formal Analysis of Normative Conflicts. *Information & Comms. Techn. Law*, 9(3):207–217, October 2000.
5. M. Fitting. *First-Order Logic and Automated Theorem Proving.* Springer-Verlag, New York, U.S.A., 1990.
6. I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Int' J. Supercomputer Applications*, 15(3):209–235, 2001.
7. A. García-Camino, J.-A. Rodríguez-Aguilar, C. Sierra, and W. Vasconcelos. A Rule-based Approach to Norm-Oriented Programming of Electronic Institutions. *ACM SIGecom Exchanges*, 5(5):33–40, January 2006.
8. A. García-Camino, J.-A. Rodriguez-Aguilar, C. Sierra, and W. W. Vasconcelos. A Distributed Architecture for Norm-Aware Agent Societies. In *DALT 2005*, volume 3904 of *LNAI*. Springer-Verlag, 2005.
9. Andrés García-Camino, Pablo Noriega, and Juan-Antonio Rodríguez-Aguilar. An Algorithm for Conflict Resolution in Regulated Compound Activities. In *Seventh Annual International Workshop Engineering Societies in the Agents World (ESAW'06)*, September 2006.
10. Joxan Jaffar and Michael J. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Progr.*, 19/20:503–581, 1994.
11. Joxan Jaffar, Michael J. Maher, Kim Marriott, and Peter J. Stuckey. The Semantics of Constraint Logic Programs. *Journal of Logic Programming*, 37(1-3):1–46, 1998.
12. M.J. Kollingbaum. *Norm-governed Practical Reasoning Agents.* PhD thesis, University of Aberdeen, 2005.
13. M.J. Kollingbaum, T.J Norman, A. Preece, and D. Sleeman. Norm Refinement: Informing the Re-negotiation of Contracts. In G. Boella, O. Boissier, E. Matson, and J. Vazquez-Salceda, editors, *ECAI 2006 Workshop on Coordination, Organization, Institutions and Norms in Agent Systems, COIN@ECAI 2006*, pages 46–51, 2006.
14. J. A. Leite, J. J. Alferes, and L. M. Pereira. Multi-Dimensional Dynamic Knowledge Representation. volume 2173 of *LNAI*. Springer-Verlag, 2001.
15. T.J. Norman, A. Preece, S. Chalmers, N.R. Jennings, M. Luck, V.D. Dang, T.D. Nguyen, V. Deora, J. Shao, W.A. Gray, and N.J. Fiddian. Agent-based Formation of Virtual Organisations. *Knowledge Based Systems*, 17:103–111, 2004.
16. O. Pacheco and J. Carmo. 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, 2003.
17. A. Ross. *On Law and Justice.* Stevens & Sons, 1958.

18. Giovanni Sartor. The Structure of Norm Conditions and Nonmonotonic Reasoning in Law. In *Proceedings of the 3rd international conference on Artificial intelligence and law ICAIL '91*, 1991.
19. Giovanni Sartor. A simple computational model for nonmonotonic and adversarial legal reasoning. In *Proceedings of the 4th international conference on Artificial intelligence and law ICAIL '93*, 1993.
20. M. Sergot. A Computational Theory of Normative Positions. *ACM Trans. Comput. Logic*, 2(4):581–622, 2001.
21. Swedish Institute of Computer Science. *SICStus Prolog*, 2005. `http://www.sics.se/isl/sicstuswww/site/index.html`, viewed on 10 Feb 2005 at 18.16 GMT.
22. W.W. Vasconcelos, M.J. Kollingbaum, T.J. Norman, and A. García-Camino. Resolving Conflict and Inconsistency in Norm-Regulated Virtual Organizations. In *Proceedings of AAMAS 2007*, 2007.
23. Stephanie M. White. Requirements for Distributed Mission-Critical Decision Support Systems. In *Procs 13th Annual IEEE Int'l Symp. & Workshop on Eng. of Computer-Based Systs. (ECBS'06)*, 2006.