Contract Formation through Preemptive Normative Conflict Resolution*

Wamberto W. Vasconcelos† and Timothy J. Norman‡

Dept. of Computing Science, University of Aberdeen, AB24 3UE, United Kingdom
†w.w.vasconcelos@abdn.ac.uk, ‡t.j.norman@abdn.ac.uk

Abstract. We explore a rule-based formalisation for contracts: the rules capture conditional norms, that is, they describe situations arising during the enactment of a multi-agent system, and norms that arise from these situations. However, such rules may establish conflicting norms, that is, norms which simultaneously prohibit and oblige (or prohibit and permit) agents to perform particular actions. We propose to use a mechanism to detect and resolve normative conflicts in a preemptive fashion: these mechanisms are used to analyse a contract and suggest "amendments" to the clauses of the contract. These amendments narrow down the scope of influence of norms and avoid normative conflicts. Agents propose rules and their amendments, leading to a contract in which no conflicts may arise.

1 Introduction

We explore a rule-based formalisation for contracts: the rules capture conditional norms, that is, they describe situations arising during the enactment of a multi-agent system (MAS), and norms that arise from these situations. However, such rules may establish conflicting norms, that is, norms which simultaneously prohibit and oblige (or prohibit and permit) agents to perform particular actions. We propose to use a mechanism to detect and resolve normative conflicts in a preemptive fashion: these mechanisms are used to analyse a contract and suggest "amendments" to the clauses of the contract. These amendments narrow down the scope of influence of norms and avoid normative conflicts.

We envisage a scenario in which agents propose rules which will make up a contract. Agents, however, may already be committed to existing contracts when they are negotiating the terms of a new contract. Furthermore, these agents may not want to divulge the terms of the contracts they have established, that is, they may not want to justify why they need to propose amendments to a contract.

* 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 authorised to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

Dagstuhl Seminar Proceedings 09121
Normative Multi-Agent Systems
http://drops.dagstuhl.de/opus/volltexte/2009/1920

The structure of this paper is as follows. In Section 2 we introduce norm-governed multi-agent systems, also presenting our account of norms and their formal underpinnings. Section 3 formally presents the syntax and semantics of contracts as a set of rules; additionally that section provides a computational model for contract enactments. In Section 4 we present mechanisms to detect and resolve normative conflicts, using unification and constraint satisfaction techniques. In Section 5 we introduce our preemptive approach to contract formation, whereby agents exchange messages with contract clauses and amendments to these. We compare our approach with related work in Section 6 and conclude in Section 7, where we also give directions for future work.

2 Norm-Governed Multi-Agent Systems

The design of complex multi-agent systems 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.*, [4, 17] – 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.

The building blocks of our formalism are *terms*:

Definition 1. *A term, denoted as τ, is any variable x, y, z (with or without subscripts) or any construct $f^n(\tau_1, \ldots, \tau_n)$, where f^n is an n-ary function symbol and $\tau_1, \ldots, \tau_n$ are terms.*

Terms f^0 stand for *constants* and will be denoted as a, b, c (with or without subscripts). We shall also make use of numbers and arithmetic functions to build our terms; arithmetic functions may appear infix, following their usual conventions. 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. Some examples of terms are *Price* (a variable) and $send(a, B, inform(c))$ (a function).

We also define *atomic formulae*:

Definition 2. *An atomic formula, denoted as φ, is any construct $p^n(\tau_1, \ldots, \tau_n)$, where p^n is an n-ary predicate symbol and $\tau_1, \ldots, \tau_n$ are terms.*

When the context makes it clear what n is we can drop it. p^0 stands for propositions. We shall employ arithmetic relations (*e.g.*, $=$, $\neq$, and so on) as predicate symbols, and these will appear in their usual infix notation. We also make use

of atomic formulae built with arithmetic relations to represent *constraints* on variables – these atomic formulae have a special status, as we explain below. We give a definition of our constraints, a subset of atomic formulae:

Definition 3. *A constraint γ is an infix binary atomic formula $\tau \lhd \tau'$, where $\lhd$ is any of the symbols $=, \neq, >, \geq, <,$ or $\leq$.*

We shall denote a possibly empty set of constraints as $\Gamma = \{\gamma_0, \ldots, \gamma_n\}$ and it stands for a *conjunction* of the constraints, that is, $\bigwedge_{i=0}^{n} \gamma_i$. 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 $\{10 \leq X \leq 45\}$.

We need an account of those actions performed by agents:

Definition 4. *An action tuple is $\langle a{:}r, \bar{\varphi} \rangle$ 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 $\bar{\varphi}$*

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.

2.1 A Representation for Norms

In this section we introduce our representation of norms. We extend our previous work [22, 23], adopting the notation of [17] for specifying norms, complementing it with *constraints* [9]. Constraints are used to further *refine* the scope of influence of norms on actions.

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

Definition 5. *A norm ω is any construct*

- *$\mathsf{O}_{\alpha:\rho}\varphi \circ \Gamma$ (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, φ is a first-order atomic formula and Γ is a possibly empty set of constraints.

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 ω. For instance, $\mathsf{P}_{A:R}\mathit{deploy}(X, b, c)$ 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 [5, 15].

We propose to formally represent from a global perspective the normative positions [19] of all agents taking part in a virtual society. By "normative position" we mean the "social burden" associated with individuals [6], that is, their obligations, permissions and prohibitions.

2.2 Substitutions, Unification and Constraint Satisfaction

We use first-order unification [5] and constraint satisfaction [10] 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 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. $(\mathsf{X}_{\alpha:\rho}\varphi \circ \Gamma) \cdot \sigma = (\mathsf{X}_{(\alpha\cdot\sigma):(\rho\cdot\sigma)}(\varphi \cdot \sigma) \circ (\Gamma \cdot \sigma))$.

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 [5]; 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 [9, 10] 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 fulfills 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*.*

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 [21] 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.

2.3 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

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

Fig. 1. Check if Action is within Influence of a Norm

the format of Def. 4), is within the influence of a norm ω (in the format of Def. 5), 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.

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 norm-compliant. The agents will thus violate a norm, and will be subject to sanctions.

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 [6].

3 Contracts as Rules for Managing Enactment States

In this section we introduce a rule-based language for the explicit management of events generated by agents and the effects they cause – we introduced alternative versions of this formalism in [7, 8]: rules depict how norms should be inserted and removed as a result of agents' actions. A contract is a set of such rules, specifying how agents' normative positions change as a result of their actions.

For our computational model we propose a global account of all actions performed, as well as all norms which currently hold. We make use of the set Δ to store action tuples and norms – it represents a *trace* or a history of the enactment of a society of agents from a global point of view:

Definition 9. *A global enactment state Δ is a finite, possibly empty, set of action tuples $\langle a\!:\!r, \bar{\varphi}\rangle$ and norms ω.*

A global enactment state Δ can be "sliced" into many partial states $\Delta_a = \{\langle a\!:\!r, \bar{\varphi}\rangle \in \Delta \,|\, a \in \mathit{Agents}\}$ containing all actions of a specific agent a. Similarly, we could have partial states $\Delta_r = \{\langle a : r, \bar{\varphi}\rangle \in \Delta \,|\, r \in \mathit{Roles}\}$, representing the global state Δ "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[1].

Figure 2 depicts how our computational model works. An initial enactment state Δ_0 (possibly empty) is offered (represented by "$\Rrightarrow$") to a set of agents $(ag_1, \ldots, ag_n)$. These agents can add their events $(\Xi_1^0, \ldots, \Xi_n^0)$ to the state of

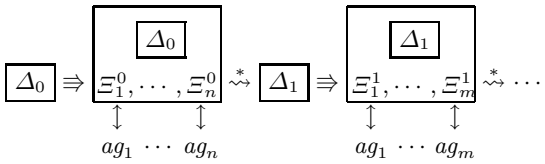

Fig. 2. Semantics as a Sequence of Δ's

affairs (via "$\updownarrow$"). Ξ_i^j is the (possibly empty) set of events added by agent i at state of affairs Δ_j. After an established amount of time, we perform an exhaustive application of rules (denoted by "$\overset{*}{\rightsquigarrow}$") to the enactment state $\Delta_0 \cup \Xi_1^0 \cup \cdots \cup \Xi_n^1$, yielding a new enactment state Δ_1. This new state will, on its turn, be offered to the agents for them to add their events, and the same process will go on.

3.1 A Rule Language for Managing Normative Positions

Our rules are constructs of the form $\mathit{LHS} \rightsquigarrow \mathit{RHS}$, where LHS contains a representation of parts of the current enactment state which, if they hold, will cause

[1] In [6] we present a distributed architecture for electronic institutions [4], 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).

the rule to be triggered. *RHS* describes the updates to the current enactment state, yielding the next enactment state:

Definition 10. *A rule R is defined by the following grammar:*

$$\begin{aligned} R &::= LHS \rightsquigarrow RHS \\ LHS &::= LHS \wedge LHS \mid \neg LHS \mid Action \mid \omega \mid \gamma \\ RHS &::= RHS \wedge RHS \mid \oplus \omega \mid \ominus \omega \end{aligned}$$

Intuitively, the left-hand side *LHS* describes the conditions the current enactment state ought to have for the rule to apply. The right-hand side *RHS* describes the updates to be performed to the current enactment state, yielding the next enactment state.

3.2 Semantics of Rules

As suggested in Figure 2, we define the semantics of our rules as a relationship between the current enactment state and the next enactment state. In this section we define this relationship.

We first define the semantics of the *LHS* of a rule, that is, how a rule is triggered:

Definition 11. $\mathbf{s}_l(\Delta, LHS, \sigma)$ *holds between an enactment state* Δ*, the left-hand side of a rule LHS and a substitution* σ *depending on the format of LHS:*

1. $\mathbf{s}_l(\Delta, LHS \wedge LHS', \sigma)$ *holds iff* $\mathbf{s}_l(\Delta, LHS, \sigma')$ *and* $\mathbf{s}_l(\Delta, \mathrm{LHS}' \cdot \sigma', \sigma'')$ *hold,* $\sigma = \sigma' \cup \sigma''$.
2. $\mathbf{s}_l(\Delta, \neg LHS, \sigma)$ *holds iff* $\mathbf{s}_l(\Delta, LHS, \sigma)$ *does not hold.*
3. $\mathbf{s}_l(\Delta, Action, \sigma)$ *holds iff* $Action \cdot \sigma \in \Delta$.
4. $\mathbf{s}_l(\Delta, \omega, \sigma)$ *holds iff* $\omega \cdot \sigma \in \Delta$.
5. $\mathbf{s}_l(\Delta, \gamma, \sigma)$ *holds iff satisfy*$(\{\gamma \cdot \sigma\})$.

Case 1 depict the semantics of conjunctions and how their individual substitutions are combined. Case 2 introduces the negation by failure: a negated action is true if, and only if, it has not taken place, that is, it is not found in the enactment state. Case 3 holds when an action is found in the enactment state. Case 3 holds when a norm is found in the enactment state. Case 5 holds if a constraint is satisfiable, after applying a substitution σ to it.

We now define the semantics of the *RHS* of a rule:

Definition 12. *Relation* $\mathbf{s}_r(\Delta, RHS, \Delta')$ *mapping an enactment state* Δ*, the right-hand side of a rule RHS and a new enactment state* Δ' *is defined as:*

1. $\mathbf{s}_r(\Delta, RHS \wedge RHS', \Delta'')$ *holds iff* $\mathbf{s}_r(\Delta, RHS, \Delta')$ *and* $\mathbf{s}_r(\Delta', RHS', \Delta'')$ *hold.*
2. $\mathbf{s}_r(\Delta, \oplus\omega, \Delta \cup \{\omega\})$ *holds.*
3. $\mathbf{s}_r(\Delta, \ominus\omega, \Delta \setminus \{\omega\})$ *holds.*

Case 1 decomposes a conjunction and builds the new state by merging the partial states of each update. Case 2 caters for the insertion of norms and case 3 defines how a norm is deleted.

Our rules are *exhaustively* applied on the enactment states thus considering all matching atomic formulae. We thus need relationship $\mathbf{s}_l^*(\Delta, LHS, \Sigma)$ which obtains in $\Sigma = \{\sigma_0, \ldots, \sigma_n\}$ *all* possible matches of the left-hand side of a rule:

Definition 13. $\mathbf{s}_l^*(\Delta, LHS, \Sigma)$ *holds, iff* $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ *is the largest non-empty set such that* $\mathbf{s}_l(\Delta, LHS, \sigma_i), 1 \leq i \leq n$, *holds.*

In the complete definition of the rule system, we define the semantics of our rules as relationships between enactment states: rules map an existing enactment state to a new enactment state. We adopt the usual semantics of production rules [14], that is, we exhaustively apply each rule by matching its *LHS* against the current state and use the values of variables obtained in this match to instantiate *RHS*.

3.3 An Interpreter for Contracts

The semantics above provides a basis for an interpreter for rules, shown in Fig. 3 as a logic program, interspersed with built-in Prolog predicates; for easy referencing, we show each clause with a number on its left. Clause 1 contains the top

1. $\mathbf{s}^*(\Delta, Rs, \Delta') \leftarrow$
   $\texttt{findall}(\langle RHS, \Sigma\rangle, (\texttt{member}((LHS \rightsquigarrow RHS), Rs), \mathbf{s}_l^*(\Delta, LHS, \Sigma)), RHSs),$
   $\mathbf{s}_r'(\Delta, RHSs, \Delta')$

2. $\mathbf{s}_l^*(\Delta, LHS, \Sigma) \leftarrow \texttt{findall}(\sigma, \mathbf{s}_l(\Delta, LHS, \sigma), \Sigma)$
3. $\mathbf{s}_l(\Delta, (Action \wedge LHS), \sigma_1 \cup \sigma_2) \leftarrow \mathbf{s}_l(\Delta, Action, \sigma_1), \mathbf{s}_l(\Delta, LHS, \sigma_2)$
4. $\mathbf{s}_l(\Delta, \neg LHS, \sigma) \leftarrow \neg\mathbf{s}_l(\Delta, LHS, \sigma)$
5. $\mathbf{s}_l(\Delta, Action, \sigma) \leftarrow \texttt{member}(Action \cdot \sigma, \Delta)$
6. $\mathbf{s}_l(\Delta, \omega, \sigma) \leftarrow \texttt{member}(\omega \cdot \sigma, \Delta)$
7. $\mathbf{s}_l(\Delta, \gamma, \sigma) \leftarrow satisfy(\{\gamma \cdot \sigma\})$

8. $\mathbf{s}_r'(\Delta, RHS, \Delta') \leftarrow$
   $\texttt{findall}(\Delta'', (\texttt{member}(\langle RHS, \Sigma\rangle, RHSs), \texttt{member}(\sigma, \Sigma), \mathbf{s}_r(\Delta, RHS \cdot \sigma, \Delta'')), AllDeltas),$
   $merge(AllDeltas, \Delta')$
9. $\mathbf{s}_r(\Delta, (U \wedge RHS), \Delta_1 \cup \Delta_2) \leftarrow \mathbf{s}_r(\Delta, U, \Delta_1), \mathbf{s}_r(\Delta, RHS, \Delta_2)$
10. $\mathbf{s}_r(\Delta, \oplus\omega, \Delta \cup \{\omega\})) \leftarrow$
11. $\mathbf{s}_r(\Delta, \ominus\omega, \Delta \setminus \{\omega\})) \leftarrow$

Fig. 3. An Interpreter for Contracts

most definition: given a Δ and a set of rules (a contract) Rs, it shows how we can obtain the next state Δ' by finding (via the built-in `findall` predicate[2]) all those rules in Rs (picked by the `member` built-in) whose *LHS* holds in Δ (checked via the auxiliary definition $\mathbf{s}_l^*$). This clause then uses the *RHS* of those rules with their respective sets of substitutions Σ as the arguments of $\mathbf{s}_r'$ to finally obtain Δ'.

Clause 2 implements $\mathbf{s}_l^*$: it finds all the different ways (represented as individual substitutions σ) that the left-hand side *LHS* of a rule can be matched in

[2] ISO Prolog built-in `findall`/3 obtains all answers to a query (2nd argument), recording the values of the 1st argument as a list stored in the 3rd argument.

an enactment state Δ – the individual σ's are stored in sets Σ of substitutions, as a result of the `findall`/3 execution. Clauses 3-7 are adaptations of Def. 11.

Clause 8 shows how $\mathbf{s}'_r$ computes the new enactment state using the current enactment state and a list *RHSs* of pairs $\langle RHS, \Sigma \rangle$ (obtained in the second body goal of clause 1): it picks out (via predicate `member`/2) each individual substitution $\sigma \in \Sigma$ and uses it in *RHS* to compute via $\mathbf{s}_r$ a partial new state Δ'' which is stored in *AllDeltas*. *AllDeltas* contains a set of partial new states and these are combined together via the *merge*/2 predicate – it joins all the partial states, removing any replicated components. Clauses 9-11 are adaptations of Def. 12.

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. 4, we show two norms over action

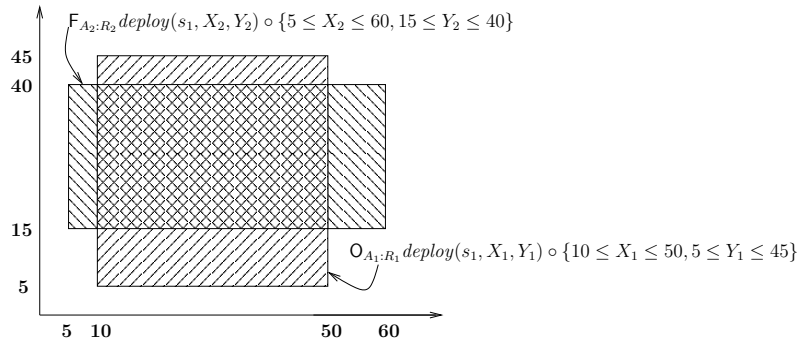

Fig. 4. Conflict Detection: Overlap in Scopes of Influence

$deploy(S, X, Y)$, 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 14. *Norms $\omega, \omega' \in \Delta$, are in conflict under substitution σ, denoted as conflict$(\omega, \omega', \sigma)$, X being O or P, iff:*

- $\omega = \mathsf{F}_{\alpha:\rho}\varphi \circ \Gamma$, $\omega' = \mathsf{X}_{\alpha':\rho'}\varphi' \circ \Gamma'$ *or*
- $\omega = \mathsf{X}_{\alpha:\rho}\varphi \circ \Gamma$, $\omega' = \mathsf{F}_{\alpha':\rho'}\varphi' \circ \Gamma'$

and the following conditions hold:

1. *unify*$(\langle \alpha, \rho, \varphi \rangle, \langle \alpha', \rho', \varphi' \rangle, \sigma)$ *and*
2. *satisfy*$((\Gamma \cup \Gamma') \cdot \sigma)$

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, and *2)* the constraints from both norms can be satisfied (taking σ under consideration).

The norm conflict of Fig. 4 is indeed captured by Definition 14. 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 ω are replaced by a fresh variable x (not occurring anywhere in ω) and a constraint $x = \tau$ is added to Γ. This is an extended form of *explicit unification* [20] and the transformation of formulae from their usual format to this extended explicit unification format can be easily automated by scanning ω 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. 5 we show the norms of Fig. 4 without the intersection between their scopes of influence[3] – the prohibition has been *curtailed*, its scope being reduced to avoid the values that

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

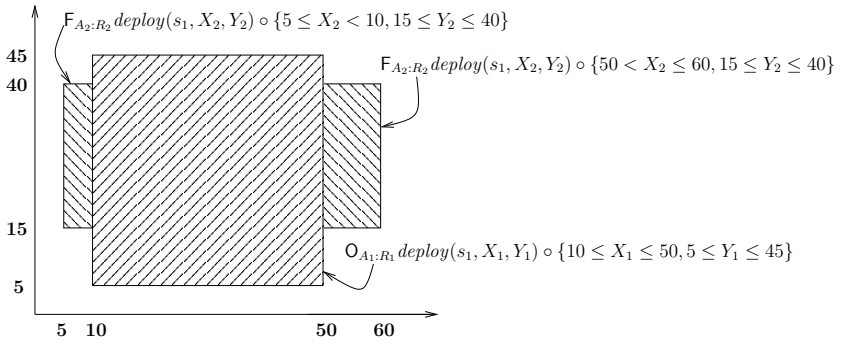

Fig. 5. Conflict Resolution: Curtailment of Scopes of Influence

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 15. *Relationship curtail$(\omega, \omega', \Omega)$, where*

- $\omega = \mathsf{X}_{\alpha:\rho}\varphi \circ \{\gamma_0, \ldots, \gamma_n\}$ *and*
- $\omega' = \mathsf{X}'_{\alpha':\rho'}\varphi' \circ \{\gamma'_0, \ldots, \gamma'_m\}$

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 $\omega^c_j = \mathsf{X}_{\alpha:\rho}\varphi \circ (\{\gamma_0, \ldots, \gamma_n\} \cup \{\neg(\gamma'_j \cdot \sigma)\})$, $0 \leq j \leq m$.*

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 $\omega = \mathsf{X}_{\alpha:\rho}\varphi \circ \{\gamma_0, \ldots, \gamma_n\}$ with the negated constraints of $\omega' = \mathsf{X}'_{\alpha':\rho'}\varphi' \circ \{\gamma'_0, \ldots, \gamma'_m\}$. 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

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

We can rewrite the last formula as

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

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

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

Although each of the $\omega_j^c, 0 \leq j \leq m$, 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. Fig. 5 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.

5 Preemptive Normative Conflict Resolution

The rules of a contract create and remove norms. When new norms are introduced, they may conflict with each other. A post-conflict approach would invoke the norm curtailment mechanism above whenever a conflict arises during the enactment of a multi-agent system which is subject to a contract. We have pursued in [23] this approach: when a new norm is added to a set of norm, it is checked for conflicts and, depending on explicit policies, either the new norm is curtailed or existing norms are curtailed. When a norm is removed, any previous curtailments it caused on other norms are *undone*, this being achieved via a "roll back"/"roll forward" mechanism: the sequence (*i.e.*, the history) of all enactment states is maintained and we roll back to the state before the norm to be removed was introduced, then skip the following state and roll forward, introducing all the norms from that point onwards.

However, this approach is computationally very expensive, as we reported in [23]. We thus suggest a *preemptive approach* whereby rules are analysed beforehand for their potential conflicts, and then the norms appearing on their right-hand sides are curtailed, thus preventing any normative conflicts in the future.

Two rules R, R' have the potential for raising a normative conflict if *i)* their *RHS*s add conflicting norms to the enactment state and *i)* their *LHS*s can be simultaneously triggered. The first check is straightforward: we can scan the *RHS*s of the rules, collect their norms, and compare them two by two. The second check, however, is much trickier as we cannot in general decide if two *LHS*s *will be* simultaneously triggered: this check would require the exhaustive

generation of all histories (*i.e.*, sequences of enactment states) and this could be prohibitively costly or, in the case of MASs which should run forever, impossible.

We address the second check in a conservative fashion: instead of checking whether the two *LHS*'s simultaneously trigger, we check if the situations they describe can possibly appear together. For instance, a if rule R has $send(Ag_1 : R_1, Ag_2 : R_2, offer(X))$ on its *LHS* and rule R' has $\neg send(Ag_1 : R_1, Ag_2 : R_2, offer(X))$ on its *LHS*, then we know for sure that these rules will never trigger simultaneously.

Given two rules $R = LHS \rightsquigarrow RHS$ and $R' = LHS' \rightsquigarrow RHS'$, we propose the $compatible(LHS, LHS')$ predicate to check if there could be an enactment state in which both *LHS* and LHS' holds. This predicate works by incrementally building an enactment state, adding to it all the actions, norms and constraints from *LHS* that are checked for, then extending the enactment state with the actions, norms and constraints that are checked for in LHS'. In this approach, we also add negated actions and norms to the state being built, so as to check if the state has a pair $\langle Action, \neg Action \rangle$, $\langle \omega, \neg\omega \rangle$ or $\langle \gamma, \neg\gamma \rangle$; if any of these appear in the state, then the rules are not compatible.

We not formally define the potential conflicts between two rules of a contract:

Definition 16. *Two rules $R = LHS \rightsquigarrow RHS$ and $R' = LHS' \rightsquigarrow RHS'$ are potentially in conflict, denoted as $conflict^r_p(R, R', \sigma)$, iff $compatible(LHS, LHS')$ holds, $\oplus\omega$ occurs in RHS, $\oplus\omega'$ occurs in RHS', and $conflict(\omega, \omega', \sigma)$.*

For the sake of simplicity, we assume in this paper that individual rules do not add conflicting norms. However, the mechanism we describe below could also be used to help engineers design individual rules, automatically spotting conflicts and suggesting changes to the rules.

5.1 Contract Formation via Preemptive Conflict Resolution

If two rules are found to be potentially in conflict, then their norm conflict(s) can be preemptively resolved by having the norms being added on their right-hand side curtailed using the mechanism presented above.

In this paper, we address the scenario in which two or more agents attempt to form a contract free from potential normative conflicts. The agents may have their own private contract(s) which they will need to take into account when forging new contracts. We consider two possible scenarios, explained below.

In the first scenario, an initiator agent ag_1 sends a proposal to another agent ag_2, consisting of a single rule R to become part of a contract between ag_1 and ag_2. Agent ag_2, the contacted agent, receives the rule and checks it against any of its current norms as well as rules of other contracts it has forged previously. Agent ag_2 then sends back a set $\mathcal{R}^c = \{R^c_1, \ldots, R^c_n\}$ of alternative versions of rule R, in which some of its added norms have been curtailed. The proposing agent ag_1 then chooses one of the rules from $\mathcal{R}^c$ and sends a message to ag_2 to inform its choice. This protocol is repeated again until the agents have a complete contract.

The other scenario is similar to the previous one, however, in addition to the set $\mathcal{R}^c = \{R_1^c, \ldots, R_n^c\}$ of alternative versions of the proposed rule, agent ag_2 also provides a *rationale* or justification for the suggestions. This rationale is in the form of ag_2's rules which have potential conflicts with R. When ag_1 receives the set $\mathcal{R}^c$ with the justifications, then (as in the previous scenario) it can accept the suggestions or, more interestingly, ag_1 may, on its turn, propose changes in the rules ag_2 used as rationale. This scenario may lead to longer interactions through which a new contract is forged via the revision of existing contracts. The revision of existing contracts use the same mechanism described here.

We present in Fig. 6 an algorithm which allows agents to analyse a proposed set of rules $\mathcal{R}$ with respect to another (pre-existing) set of rules $\mathcal{R}'$, providing a

algorithm $preempt(\mathcal{R}, \mathcal{R}', \mathcal{R}^c)$
input a proposed set of new rules $\mathcal{R}$, and a set of old rules $\mathcal{R}'$
output a revised set of proposed rules $\mathcal{R}^c$

```
1  begin
2    R^c ← ∅
3    for each R' ∈ R', R' = LHS' ⇝ RHS', do
4    begin
5      conflict_flag ← false
6      R^t ← ∅
7      for each R ∈ R, R = LHS ⇝ RHS, do
8        if conflict_p^r(R, R', σ) then
9        begin
10         conflict_flag ← true
11         for each ⊕ω' ∈ {RHS'} do
12           for each ⊕ω ∈ {RHS} do
13           begin
14             curtail(ω, ω', Ω)
15             for each ω^c ∈ Ω do
16             begin
17               {RHS^t} ← {RHS} \ {⊕ω} ∪ {⊕ω^c}
18               R^t ← R^t ∪ {LHS ⇝ RHS^t}
19             end
20           end
21       end
22     if ¬conflict_flag then
23     begin
24       R^c ← R^c ∪ {R}
25       R ← R \ {R}
26     end
27     else
28       R ← R \ {R} ∪ R^t
29   end
30 end
```

Fig. 6. Algorithm for Preemptive Contract Formation

set of recommended changes $\mathcal{R}^c$ to $\mathcal{R}$. These changes will guarantee that there will be no normative conflicts when the agent takes part in enactments of multi-agent systems regulated by the contracts $\mathcal{R}^c$ and $\mathcal{R}'$

In the algorithm we make use of $\{RHS\}$ to refer to the set with all the components of the right-hand side of the rule. More formally, we have:

Definition 17. *If* $RHS = \otimes\omega_1 \wedge \cdots \wedge \otimes\omega_n$ *(where* $\otimes$ *is either* $\oplus$ *or* $\ominus$*) then* $\{RHS\} = \{\otimes\omega_1, \ldots, \otimes\omega_n\}$.

The algorithm of Fig. 6 works by comparing each rule from $\mathcal{R}'$ with the candidate rules of $\mathcal{R}$. The algorithm makes use of a temporary set of changed

rules $\mathcal{R}^t$ which is initialised to $\emptyset$, the empty set, in line 6 at the start of each loop with $R \in \mathcal{R}'$. The algorithm also makes use of a flag *conflict_flag* which is set true if there is a potential conflict between any rule $R' \in \mathcal{R}'$, that is, a rule from the set of old rules, and $R \in \mathcal{R}$ a new candidate rule – the flag is initialised to false in line 5 (with each new rule $R' \in \mathcal{R}'$), and switched to **true** in line 10, when a potential conflict is detected.

Loop 3–29 goes through the old rules $R' \in \mathcal{R}'$ and compares each of them with the new rules $R \in \mathcal{R}$ (loop 7–21). For each pair R, R' (respectively members from sets $\mathcal{R}'$ and $\mathcal{R}$), the algorithm checks for potential conflicts (line 8). If there is a potential conflict (cf. Def 16), lines 11-20 scan the *RHS* of both rules, obtaining a set Ω (line 14; cf. with Def. 14) which is the result of the curtailment of norm ω (from a new rule $R \in \mathcal{R}$) with respect to ω' (from an old rule $R' \in \mathcal{R}'$).

For each curtailed norm $\omega^c \in \Omega$ (lines 15–19), the algorithm creates an alternative *RHS*, replacing the old $\oplus\omega$ with $\oplus\omega^c$ (line 17). Line 18 updates the temporary set $\mathcal{R}^t$ of curtailed rules, adding the new rules obtained by replacing $\oplus\omega$ with $\oplus\omega^c$. Lines 22–28, executed after the loop of lines 7–21, update the set $\mathcal{R}^c$ of rules free from potential conflicts. Line 24 adds R to $\mathcal{R}^c$, since it is free from potential conflicts, and line 25 removes R from the set $\mathcal{R}$; if however, there has been a conflict, the set $\mathcal{R}$ should be updated, removing the conflicting R from it, and adding all the alternative formulations to R assembled in $\mathcal{R}^t$.

For the algorithm to work we make two assumptions. Firstly, we assume that the rules in $\mathcal{R}$ do not add norms with conflicts in their *RHS*s. Secondly, we assume that the rules in $\mathcal{R}$ do not have potential conflicts among themselves. The second assumption can be accommodated in a realistic setting if rules (possibly conflicting) are submitted one at a time to the algorithm; the first assumption requires the adaptation of the algorithm to address the design of rules, interleaving design with verification, a topic we elaborate further when we discuss future work below.

6 Related Work

In this section we refer to related work addressing aspects of contracts and normative systems.

A closely related work is that of [16]. The framework proposed in that paper includes contract specification, negotiation and monitoring, as well as the appropriate agent architecture to handle these aspects. However, in that paper issues of contract formation are not explored.

The work in [13] presents legal issues and surveys efforts at standardising contracts for electronic commerce. However, that paper does not propose a formal representation for contracts, nor does it investigate how contracts can be forged in an interactive way.

There are various other formulations for norms in the literature, as well as different ways to represent clauses of contracts using norms. In [8] we compare a rule-based formalism (notably more sophisticated than the one presented in this paper) with a number of alternative approaches. We show that it is possible to

capture various normative phenomena with a rule-based formalism. Moreover, the rule-based formulation proves to be more compact, elegant, and intuitive than other formulations.

Our work is not concerned with contract negotiation based in game theoretical aspects, as explored in [2]. However, it has not escaped our attention that [2] employs a rule-based formalism, although their notion of norms is quite different from the one presented here; their conflicts are solved by means of priorities, which are used to choose a course of action.

Our approach to norm conflict detection and resolution can be contrasted with the work described in [11, 12]: 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. It is not clear how constraints in the norms of [11, 12] 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).

The work described in [3] 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 [18], *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.

7 Summary, Conclusions and Future Work

We presented formal means to represent norms, that is, prohibitions, permissions, and obligations, and how these can be combined with a rule-based formalism to specify contracts. The left-hand side of our rule describe the circumstances which ought to arise for a norm to be revoked (removed) or introduced; the right-hand side of our rules specify which norms are to be revoked or introduced; we provided a simple semantics for our norms and rules.

Our norm representation uses constraints: these allow for a fine-grained control of the scope of the norm, that is, the values of the variables the norm refers to. These constraints are also useful when conflicts arise: we propose the resolution of normative conflicts via the careful manipulation of constraints. We provide means to detect normative conflicts and how to resolve them, and use these to propose a preemptive approach to contract formation, whereby agents exchange rules of a contract, checking these against any existing norms or other

previously forged contracts. Our approach can be said to be preemptive because normative conflicts are considered *before* the contract is enacted and hence before any actual normative conflict arises.

We are exploring the proposed preemptive approach within the context of the ITA research project[4]. More specifically, we want to support coalition of human and software agents from disparate organisations (hence with different degrees of loyalty and willingness to share information and assets) to agree on the terms of a mission.

We want to adapt and extend the rule-based approach presented in this paper, using instead a logical approach. We envisage the clauses of a contract represented as formulae of a decidable fragment of first-order logic; in this approach a contract would be interpreted as a logical theory. Normative conflict can be detected via the reasoning mechanism of the logic, and the manipulation of constraints could still be used to resolve the conflicts.

References

1. K. R. Apt. *From Logic Programming to Prolog.* Prentice-Hall, U.K., 1997.
2. G. Boella and L. van der Torre. A Game Theoretic Approach to Contracts in Multiagent Systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 36(1):68–79, Jan. 2006.
3. 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, Oct. 2000.
4. M. Esteva. *Electronic Institutions: from Specification to Development.* PhD thesis, Universitat Politècnica de Catalunya (UPC), 2003. IIIA monography Vol. 19.
5. M. Fitting. *First-Order Logic and Automated Theorem Proving.* Springer-Verlag, New York, U.S.A., 1990.
6. A. García-Camino, J.-A. Rodríguez-Aguilar, C. Sierra, and W. W. Vasconcelos. A Distributed Architecture for Norm-Aware Agent Societies. In M. Baldoni, U. Endriss, A. Omicini, and P. Torroni, editors, *Procs. of the 3rd Int'l Worskhop on Declarative Agent Languages and Technologies (DALT 2005), Selected and Revised Papers*, volume 3904 of *Lecture Notes in Computer Science*, pages 89–105. Springer-Verlag, Utrecht, The Netherlands, July 25, 2005, 2006.
7. A. García-Camino, J.-A. Rodríguez-Aguilar, C. Sierra, and W. W. Vasconcelos. A Rule-based Approach to Norm-Oriented Programming of Electronic Institutions. *ACM SIGecom Exchanges*, 5(5):33–40, Jan. 2006.
8. A. García-Camino, J.-A. Rodríguez-Aguilar, C. Sierra, and W. W. Vasconcelos. Constraint Rule-Based Programming of Norms for Electronic Institutions. *Journal of Autonomous Agents & Multiagent Systems*, 18(1):186–217, Feb. 2009.
9. J. Jaffar and M. J. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Progr.*, 19/20:503–581, 1994.
10. J. Jaffar, M. J. Maher, K. Marriott, and P. J. Stuckey. The Semantics of Constraint Logic Programs. *Journal of Logic Programming*, 37(1-3):1–46, 1998.

[4] The International Technology Alliance (ITA) is a US/UK consortium of academic, military and industrial partners, pursuing research in technologies for knowledge management, communication and coordination among members of coalitions engaged in joint operations. Details about the ITA consortium are available at `http://www.usukita.org/`

11. L. Kagal and T. Finin. Modeling Communicative Behavior Using Permissions and Obligations. In *Lecture Notes in Computer Science*, volume 3396, pages 120–133, 2005.
12. L. Kagal and T. Finin. Modeling Conversation Policies using Permissions and Obligations. *Journal of Autonomous Agents & Multiagent Systems*, 14(2):187–206, Apr. 2007.
13. I. R. Kerr. Ensuring the Success of Contract Formation in Agent-Mediated Electronic Commerce. *Electronic Commerce Research*, 1(1-2):183–202, 2001.
14. B. Kramer and J. Mylopoulos. Knowledge Representation. In S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, volume 1, pages 743–759. John Wiley & Sons, 1992.
15. Z. Manna. *Mathematical Theory of Computation*. McGraw-Hill Kogakusha, Ltd., Tokio, Japan, 1974.
16. F. R. Meneguzzi, S. Miles, M. Luck, C. Holt, M. Smith, N. Oren, N. Faci, M. Kollingbaum, and S. Modgil. Electronic Contracting in Aircraft Aftercare: A Case Study. In M. Berger, B. Burg, and S. Nishiyama, editors, *Procs. 7th Int'l Joint Conf. on Autonomous Agents & Multiagent Systems (AAMAS 2008), Industry and Applications Track*, pages 63–70, Estorial, Portugal, May 2008. IFAAMAS.
17. 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.
18. A. Ross. *On Law and Justice*. Stevens & Sons, 1958.
19. M. Sergot. A Computational Theory of Normative Positions. *ACM Trans. Comput. Logic*, 2(4):581–622, 2001.
20. L. Shapiro and E. Y. Sterling. *The Art of Prolog: Advanced Programming Techniques*. The MIT Press, April 1994.
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, and T. J. Norman. Resolving Conflict and Inconsistency in Norm-Regulated Virtual Organizations. In *Procs. 6th Int'l Joint Conf. on Autonomous Agents & Multiagent Systems (AAMAS 2007)*, pages 632–639, Hawai'i, U.S.A., May 2007. IFAAMAS.
23. W. W. Vasconcelos, M. J. Kollingbaum, and T. J. Norman. Normative Conflict Resolution in Multi-Agent Systems. *Journal of Autonomous Agents & Multiagent Systems*, 2009. Volume and number to be confirmed. Available on-line at `http://www.springerlink.com/content/024242p7775530k6/`.