# A Rule-based Approach to Norm-Oriented Programming of Electronic Institutions

A. García-Camino, J.A. Rodríguez-Aguilar and C. Sierra
IIIA-CSIC, Spain
{andres,jar,sierra}@iiia.csic.es
and
W. Vasconcelos
University of Aberdeen, United Kingdom
wvasconcelos@acm.org

Norms constitute a powerful coordination mechanism among heterogeneous agents. We propose means to specify and explicitly manage the normative positions of agents (permissions, prohibitions and obligations), with which distinct deontic notions and their relationships can be captured. Our rule-based formalism includes constraints for more expressiveness and precision and allows the norm-oriented programming of electronic institutions: normative aspects are given a precise computational interpretation. Our formalism has been conceived as a machine language to which other higher-level normative languages can be mapped, allowing their execution, as we illustrate with a selection of examples from the literature.

Categories and Subject Descriptors: I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems—*Law*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multi-agent systems*

General Terms: Languages

Additional Key Words and Phrases: Norms, electronic institutions, programming, auctions

## 1. INTRODUCTION

A major challenge in multi-agent system (MAS) research is the design and implementation of *open* multi-agent systems in which coordination must be achieved among agents defined with different languages by several designers who may not trust each other [Jennings et al. 1998]. Norms can be used for this purpose as a means to regulate the observable behaviour of agents as they interact in pursuit of their goals [Wooldridge 2002; Axelrod 1997; Dignum 1999; López y López 2003]. There is a wealth of socio-philosophical and logic-theoretical literature on the subject of norms (*e.g.*, [Sergot 2001; Shoham and Tennenholtz 1995]), and, more recently, much attention is being paid to more pragmatic and implementational aspects of norms, that is, how norms can be given a computational in-

This work was partially funded by the Spanish Science and Technology Ministry as part of the Web-i-2 project (TIC-2003-08763-C02-00). Garcia-Camino enjoys an I3P grant from the Spanish Council for Scientific Research (CSIC).

terpretation and how norms can be factored in in the design and execution of MASs (e.g. [Artikis et al. 2005; García-Camino et al. 005a; García-Camino et al. 005b]).

A normative position [Sergot 2001] is the "social burden" associated with individual agents, that is, their obligations, permissions and prohibitions. Depending on what agents do, their normative positions may change – for instance, permissions/prohibitions can be revoked or obligations, once fulfilled, may be removed. Ideally, norms, once captured via some suitable formalism, should be directly executed, thus realising a computational, normative environment wherein agents interact. This is what we mean by *norm-oriented programming*. We try to make headway along this direction by introducing an executable language to specify agents' *normative positions* and manage their changes as agents interact via speech acts [Searle 1969].

In this paper we present a language that acts as a "machine language" for norms on top of which higher-level normative languages can be accommodated. This language can represent distinct flavours of deontic notions and relationships. Although our language is rule-based, we achieve greater flexibility, expressiveness and precision than production systems by allowing constraints to be part of our rules and states of affairs. In this way, normative positions can be further refined. For instance, picture a selling agent that is obliged to deliver a good satisfying some quality requirements before a deadline. Notice that both the quality requirements and the delivery deadline can be regarded as constraints that must be considered as part of the obligations. Thus, when the agent delivers the good satisfying all the constraints, we should regard the obligation as fulfilled. Notice too that since the deadline might eventually be changed, we also require the capability of modifying constraints at run-time. Hence, constraints are considered as first-class citizens in our language.

Although in this paper we restrict to a particular class of MASs, namely electronic institutions [Esteva 2003], our work sets the foundations to specify and implement open regulated MASs via norms.

The structure of this paper is as follows. In the next section we present desirable properties of normative languages. In section 3 we propose a simple normative language that covers all these requirements along with a sketch of an implementation of an interpreter. Section 4 summarises electronic institutions and explains how we capture normative positions of participating agents. We put our language to use by specifying the Dutch Auction protocol in section 5. Finally, we draw conclusions and outline future work in section 6.

## 2. DESIDERATA FOR NORM-ORIENTED MAS PROGRAMMING

Our main goal is to produce a language that supports the specification of coordination mechanisms in multi-agent systems by means of norms. For this purpose, we identify below the desirable features we expect in candidate languages.

***Explicit management of normative positions***. As a result of agents' observable, social interactions, their normative positions [Sergot 2001] change. Hence, the first requirement of our language is to support the *explicit management* of agents' normative positions.

***General purpose***. We require that our language captures different deontic notions along with their relationships. In other words, the language must be of *general purpose* so that it helps MAS designers to encode any axiomatisation, and thus specify the widest range of normative systems as possible.

***Pragmatic***. In a sense, we pursue a "machine language" for norms on top of which higher-level languages may be accommodated. Along this direction, and from a language

designer's point of view, it is fundamental to identify the *norm patterns* (*e.g.*, conditional obligation, time-based permissions and prohibitions, continuous obligation, and so on) in the literature to ensure that the language supports their encoding[1]. In this way, not only shall we be guaranteeing the expressiveness of our language, but also addressing pragmatic concerns by providing *design patterns* to guide and ease MAS design.

***Declarative***. In order to ease MAS programming, we shall also require our language to be *declarative*, with an implicit execution mechanism to reduce the number of issues designers ought to concentrate on. As an additional benefit, we expect its declarative nature to facilitate verification of properties of the specifications.

## 3. A RULE LANGUAGE FOR NORMS

The building blocks of our language are first-order terms (denoted as $\tau$) and implicitly, universally quantified atomic formulae (denoted as $\alpha$) without free variables . We shall make use of numbers and arithmetic functions to build terms; arithmetic functions may appear infix, following their usual conventions[2]. We also employ arithmetic relations (*e.g.*, $=$, $\neq$, and so on) as predicate symbols, and these will appear in their usual infix notation with their usual meaning. Atomic formulae with arithmetic relations represent *constraints* on their variables and have a special status, as we explain below. We give a definition of our constraints, a subset of atomic formulae: a constraint $\gamma$ is an atomic formula of the form $\tau \lhd \tau'$, where $\lhd \in \{=, \neq, >, \geq, <, \leq\}$.

We need to differentiate ordinary atomic formula from constraints. We shall use $\alpha'$ to denote atomic formulae that are *not* constraints.

Intuitively, a state of affairs is a set of atomic formulae. As we will show below, they can store the state of the environment[3], observable agent attributes and the normative positions of agents:

A state of affairs $\Delta = \{\alpha_0, \ldots, \alpha_n\}$ is a a finite and possibly empty set of implicitly, universally quantified atomic formulae $\alpha_i, 0 \leq i \leq n, n \in I\!N$.

Our rules are constructs of the form *LHS* $\rightsquigarrow$ *RHS*, where *LHS* contains a representation of parts of the current state of affairs which, if they hold, will cause the rule to be triggered. *RHS* depicts the updates to the current state of affairs, yielding the next state of affairs:

A rule, denoted as R, is defined as:

$$
\begin{array}{rcl}
\mathsf{R} & ::= & \mathit{LHS} \rightsquigarrow \mathit{RHS} \\
\mathit{LHS} & ::= & \mathit{LHS} \wedge \mathit{LHS} \mid \neg(\mathit{LHS} \wedge \mathit{LHS}) \mid \mathsf{Lit} \\
\mathit{RHS} & ::= & \mathsf{U} \wedge \mathit{RHS} \mid \mathsf{U} \\
\mathsf{Lit} & ::= & \alpha \mid \neg\alpha \mid x = \{\alpha' | \mathit{LHS}^*\} \\
\mathsf{U} & ::= & \oplus\alpha \mid \ominus\alpha
\end{array}
$$

where $x$ is a variable name; and *LHS*$^*$ is a *LHS* without set constructors.

The Us represent the updates: they add (via operator $\oplus$) or remove (via operator $\ominus$) atomic formulae $\alpha$s. Furthermore, we make use of a special kind of term, called a *set constructor*, represented as $\{\alpha' | \mathit{LHS}^*\}$. This construct is useful when we need to refer to all $\alpha'$s in the state of affairs for which *LHS*$^*$ holds. For instance, $\{p(A, B) | A > 20 \wedge B <$

[1] This is elaborated in `http://www.iiia.csic.es/~andres/NOPLforEIs.pdf`

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

[3] We refer to the *state of the environment* as the set of atomic formulae that represent aspects of the environment in a given point in time.

$100\}$ stands for the set of atomic formulae $p(A, B)$ such that $A$ is greater than 20 and $B$ is less than 100.

We need to refer to the set of constraints that belongs to a state of affairs. We call $\Gamma = \{\gamma_0, \ldots, \gamma_n\}$ the set of all constraints in $\Delta$.

Given a state of affairs $\Delta$, relationship $constrs(\Delta, \Gamma)$ holds iff $\Gamma$ is the smallest set such that for every $\gamma \in \Delta$ then $\gamma \in \Gamma$.

In the definitions below we rely on the concept of *substitution*, that is, the set of values for variables in a computation [Fitting 1990]: a substitution $\sigma = \{x_0/\tau_0, \ldots, x_n/\tau_n\}$ is a finite and possibly empty set of pairs $x_i/\tau_i$, $0 \leq i \leq n$, $n \in I\!N$. The application of a substitution to $\alpha$s is as follows: 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. $p^n(\tau_0, \ldots, \tau_n) \cdot \sigma = p^n(\tau_0 \cdot \sigma, \ldots, \tau_n \cdot \sigma)$.

We now define the semantics of our rules as relationships between states of affairs: rules map an existing state of affairs to a new state of affairs. We adopt the usual semantics of production rules, that is, we exhaustively apply each rule by matching its *LHS* against the current state of affairs and use the values of variables obtained in this match to instantiate the *RHS* via $\mathbf{s}^*$: $\mathbf{s}^*(\Delta, \mathit{LHS} \rightsquigarrow \mathit{RHS}, \Delta')$ holds iff $\mathbf{s}_l^*(\Delta, \mathit{LHS}, \{\sigma_1, \ldots, \sigma_n\})$ and $\mathbf{s}_r(\Delta, \mathit{RHS} \cdot \sigma_i, \Delta'), 1 \leq i \leq n, n \in I\!N$, hold.

That is, two states of affairs $\Delta$ and $\Delta'$ are related by a rule $\mathit{LHS} \rightsquigarrow \mathit{RHS}$ if, and only if, we obtain all different substitutions $\{\sigma_1, \ldots, \sigma_n\}$ that make the left-hand side match $\Delta$ and apply these substitutions to *RHS* (that is, $\mathit{RHS} \cdot \sigma_i$) in order to build $\Delta'$.

Our rules are *exhaustively* applied on the state of affairs thus considering all matching atomic formulae. We thus need relationship $\mathbf{s}_l^*(\Delta, \mathit{LHS}, \Sigma)$ which obtains in $\Sigma = \{\sigma_0, \ldots, \sigma_n\}$ *all* possible matches of the left-hand side of a rule: $\mathbf{s}_l^*(\Delta, \mathit{LHS}, \Sigma)$ holds, iff $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ is the largest non-empty set such that $\mathbf{s}_l(\Delta, \mathit{LHS}, \sigma_i), 1 \leq i \leq n, n \in I\!N$, holds.

We now define the semantics of the *LHS* of a rule: $\mathbf{s}_l(\Delta, \mathit{LHS}, \sigma)$ holds between state $\Delta$, the left-hand side of a rule *LHS* and a substitution $\sigma$ depending on the format of *LHS*:

(1) $\mathbf{s}_l(\Delta, \mathit{LHS} \wedge \mathit{LHS}', \sigma)$ holds iff $\mathbf{s}_l(\Delta, \mathit{LHS}, \sigma')$ and $\mathbf{s}_l(\Delta, \mathit{LHS}', \sigma'')$ hold and $\sigma = \sigma' \cup \sigma''$.
(2) $\mathbf{s}_l(\Delta, \neg \mathit{LHS}, \sigma)$ holds iff $\mathbf{s}_l(\Delta, \mathit{LHS}, \sigma)$ does not hold.
(3) $\mathbf{s}_l(\Delta, \alpha', \sigma)$ holds iff $\alpha' \cdot \sigma \in \Delta$ and $constrs(\Delta, \Gamma)$ and $satisfiable(\Gamma \cdot \sigma)$ hold.
(4) $\mathbf{s}_l(\Delta, \gamma, \sigma)$ holds iff $constrs(\Delta, \Gamma)$ and $satisfiable((\Gamma \cup \{\gamma\}) \cdot \sigma)$ hold.
(5) $\mathbf{s}_l(\Delta, x = \{\alpha' | \mathit{LHS}^*\}, \sigma)$ holds iff $\sigma = \{x/\{\alpha' \cdot \sigma_1, \ldots, \alpha' \cdot \sigma_n\}\}$ for the largest $n \in I\!N$ such that $\mathbf{s}_l(\Delta, \alpha' \wedge \mathit{LHS}^*, \sigma_i), 1 \leq i \leq n$

Cases 1-3 depict the semantics of atomic formulae and how their individual substitutions are combined to provide the semantics for a conjunction. Case 4 formalises the semantics of our constraints when they appear on the left-hand side of a rule: we apply the substitution $\sigma$ to them (thus reflecting any values of variables given by the matchings of atomic formula), then check satisfiability of constraints.[4] Case 5 specifies the semantics for *set constructors*: $x$ is the set of atomic formulae that satisfy the conditions of the set constructor.

We now define the semantics of the *RHS* of a rule: relation $\mathbf{s}_r(\Delta, \mathit{RHS}, \Delta')$ mapping a state $\Delta$, the right-hand side of a rule *RHS* and a new state $\Delta'$ is defined as:

(1) $\mathbf{s}_r(\Delta, (\mathsf{U} \wedge \mathit{RHS}), \Delta')$ holds iff both $\mathbf{s}_r(\Delta, \mathsf{U}, \Delta_1)$ and $\mathbf{s}_r(\Delta_1, \mathit{RHS}, \Delta')$ hold.

[4] Our work builds on standard technologies for constraint solving – in particular, we have been experimenting with SICStus Prolog constraint satisfaction libraries.

(2) $\mathbf{s}_r(\Delta, \oplus\alpha', \Delta')$ holds iff $\Delta' = \Delta \cup \{\alpha'\}$.
(3) $\mathbf{s}_r(\Delta, \oplus\gamma, \Delta') = \mathbf{true}$ iff $constrs(\Delta, \Gamma)$ and $satisfiable(\Gamma \cup \{\gamma\})$ hold and $\Delta' = \Delta \cup \{\gamma\}$.
(4) $\mathbf{s}_r(\Delta, \ominus\alpha, \Delta')$ holds iff $\Delta' = \Delta \setminus \{\alpha\}$

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 atomic formulae $\alpha'$ which do not conform to the syntax of constraints. Case 3 defines how a constraint is added to a state $\Delta$: the new constraint is checked whether it can be satisfied with constraints $\Gamma$ and then it is added to $\Delta'$. Case 4 caters for the removal of atomic formulae.

We extend $\mathbf{s}^*$ to handle sets of rules: $\mathbf{s}^*(\Delta_0, \{\mathsf{R}_1, \ldots, \mathsf{R}_n\}, \Delta_n)$ holds iff $\mathbf{s}^*(\Delta_{i-1}, \mathsf{R}_i, \Delta_i), 1 \leq i \leq n$ hold. The semantics above define an infinite sequence of states $\langle \Delta_0, \Delta_1, \ldots \rangle$ if $\mathbf{s}^*(\Delta_i, \{\mathsf{R}_1, \ldots, \mathsf{R}_n\}, \Delta_{i+1})$, that is, $\Delta_{i+1}$ (obtained by applying the rules to $\Delta_i$) is used to obtain $\Delta_{i+2}$ and so on. Fig. 1 illustrates how this sequence can accommodate the intervention of agents sending/receiving messages. The diagram shows an initial state

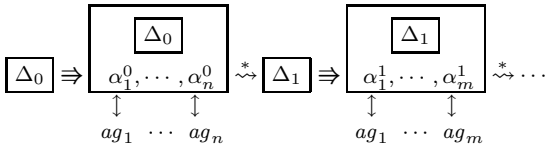


Fig. 1. Semantics as a Sequence of $\Delta$'s

$\Delta_0$ (possibly empty) that is offered (represented by "$\Rightarrow$") to a set of agents $\{ag_1, \ldots, ag_n\}$. These agents exchange messages, adding a record (via "$\updownarrow$") $\{\alpha_1^0, \ldots, \alpha_n^0\}$ of these messages to $\Delta_0$. After the agents add their utterances, then the rules are exhaustively applied (represented by "$\overset{*}{\rightsquigarrow}$") to $\Delta_0 \cup \{\alpha_1^0, \ldots, \alpha_n^0\}$. The resulting state $\Delta_1$ is, on its turn, offered to agents, and so on.

## 4. ELECTRONIC INSTITUTIONS

Our work extends *electronic institutions* (EIs) [Esteva 2003], providing them with an explicit normative layer. There are two major features in EIs – the *states* and *illocutions* (*i.e.*, messages) uttered (*i.e.*, sent) by those agents taking part in the EI. The states are connected via edges labelled with the illocutions that ought to be sent at that particular point in the EI. Another important feature in EIs are the agents' *roles*: these are labels that allow agents with the same role to be treated collectively thus helping engineers abstract away from individuals. We define below the class of illocutions we aim at – these are a special kind of term: illocutions $\mathsf{I}$ are terms $p(ag, r, ag', r', \tau, t)$ where $p$ is an illocutionary particle (*e.g.*, *inform*,*ask*); $ag, ag'$ are agent identifiers; $r, r'$ are role labels; $\tau$ is a term with the actual content of the message and $t \in I\!N$ is a time stamp. We shall refer to illocutions that may have uninstantiated (free) variables as *illocution schemes*, denoted by $\bar{\mathsf{I}}$.

Another important concept in EIs we employ here is that of a *scene*. Scenes offer means to break down larger protocols into smaller ones with specific purposes. We can uniquely refer to the point of the protocol where an illocution $\mathsf{I}$ was uttered by the pair $(s, w)$ where $s$ is a scene name and $w$ is the state from which an edge labelled with $\bar{\mathsf{I}}$ leads to another state.

We differentiate seven kinds of atomic formulae in our state of affairs $\Delta$, with the following intuitive meanings:

(1) ground formulae $oav(o, a, v)$ – object (or agent) $o$ has an attribute $a$ with value $v$.

(2) ground formulae $att(s, w, \mathsf{I})$ – an agent attempted to get illocution $\mathsf{I}$ accepted at state $w$ of scene $s$.
(3) ground formulae $utt(s, w, \mathsf{I})$ – $\mathsf{I}$ was accepted as a legal utterance at $w$ of $s$.
(4) ground formulae $ctr(s, w, t_s)$ – the execution of scene $s$ reached state $w$ at time $t_s$.
(5) $obl(s, w, \bar{\mathsf{I}})$ – $\bar{\mathsf{I}}$ ought to be uttered at $w$ of $s$.
(6) $per(s, w, \bar{\mathsf{I}})$ – $\bar{\mathsf{I}}$ is *permitted* to be uttered at $w$ of $s$.
(7) $prh(s, w, \bar{\mathsf{I}})$ – $\bar{\mathsf{I}}$ is *prohibited* at $w$ of $s$.

We only allow fully ground attributes, illocutions and state control formulae (cases 1-4 above) to be present[5]; however, in formulae 5-7 $s$ and $w$ may be variables and $\bar{\mathsf{I}}$ may contain variables. We shall use formulae 4 to represent state change in a scene in relationship with global time passing. We shall use formulae 5–7 above to represent normative positions of agents within EIs. This allow us to explicitly manage normative positions of agents in our language. We have thus addressed another requirement laid out in section 2.

## 5. EXAMPLE: THE DUTCH AUCTION

We now illustrate the pragmatics of our norm-oriented language by specifying the auction protocol for a fish market as described in [Noriega 1997], a variation of the traditional Dutch auction protocol that proceeds as follows: (1) The auctioneer chooses a good out of a lot of goods that is sorted according to the order in which sellers deliver their goods to the sellers' admitter; (2) with a chosen good, the auctioneer opens a *bidding round* by quoting offers downward from the good's starting price, previously fixed by a sellers' admitter, as long as these price quotations are above a *reserve price* previously defined by the seller; (3) for each price the auctioneer calls, several situations might arise during the open round described below.(4) The first three steps repeat until there are no more goods left.

The situations arising in step 3 are: **Multiple bids** – Several buyers submit their bids at the current price. In this case, a collision comes about, the good is not sold to any buyer, and the auctioneer restarts the round at a higher price; **One bid** – Only one buyer submits a bid at the current price. The good is sold to this buyer whenever his credit can support his bid. Otherwise, the round is restarted by the auctioneer at a higher price, the unsuccessful bidder is fined; **No bids** – No buyer submits a bid at the current price. If the reserve price has not been reached yet, the auctioneer quotes a new price obtained by decreasing the current price according to the price step. Otherwise, the auctioneer declares the good as *withdrawn* and closes the round.

### 5.1 Proposed Solution

We show our solution in Fig. 2. The rules are: **I. Multiple bids** – it obliges the auctioneer to inform the buyers, whenever a collision comes about, about the collision and to restart the bidding round at a higher price (in this case, 120% of the collision price). Notice that $X$ will hold all the utterances at scene $dutch$ and state $w_4$ issued by buyer agents that bid for an item $It$ at price $P$ at time $T_0$ after the last offer. We obtain the last offers by checking that there are no further offers whose time-stamps are greater than the time-stamp of the first one. If the number of illocutions in $X$ is greater than one, the rule fires the obligation above. **II. One bid/winner determination** – If only one bid has occurred during the current bidding round and the credit of the bidding agent is greater than or equal

[5]We allow agents to utter whatever they want (via $att$ formulae). However, the illegal utterances may be discarded and/or may cause sanctions, depending on the deontic notions we want or need to implement. The $utt$ formulae are thus *confirmations* of the $att$ formulae.

$$\left( X = \{\ \alpha_0 \mid \alpha_1 \wedge \neg(\alpha_2 \wedge T_2 > T_1) \wedge T_0 > T_1\ \} \wedge |X| > 1 \right) \rightsquigarrow \left( \oplus\alpha_3 \wedge \oplus\alpha_4 \wedge \oplus(P_2 > P * 1.2) \right)$$

$$where \begin{cases} \alpha_0 = utt(dutch, w_4, inform(A_1, buyer, Au, auct, bid(It, P), T_0)) \\ \alpha_1 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T_1)), \\ \alpha_2 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T_2)) \\ \alpha_3 = obl(dutch, w_5, inform(Au, auct, all, buyer, collision(It, P), T_2)) \\ \alpha_4 = obl(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P_2), T_3)) \end{cases} \quad \text{(I)}$$

$$\left( \begin{array}{c} X = \{\ \alpha_0 \mid \alpha_1 \wedge \neg(\alpha_2 \wedge T_2 > T_1) \wedge T_0 > T_1\ \} \wedge \\ |X| = 1 \wedge oav(A_1, credit, C) \wedge C \geq P \end{array} \right) \rightsquigarrow \left( \oplus\alpha_3 \right)$$

$$where \begin{cases} \alpha_0 = utt(dutch, w_4, inform(A_1, buyer, Au, auct, bid(It, P), T_0)) \\ \alpha_1 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T_1)), \\ \alpha_2 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T_2)) \\ \alpha_3 = obl(dutch, w_5, inform(Au, auct, all, buyer, sold(It, P, A_1), T_4)) \end{cases} \quad \text{(II)}$$

$$\left( \alpha_0 \wedge \neg(\alpha_1 \wedge T2 > T) \wedge oav(Ag, credit, C) \wedge C < P \right) \rightsquigarrow \left( \oplus\alpha_2 \right)$$

$$where \begin{cases} \alpha_0 = utt(dutch, w_3, inform(Au, auct, A, buyer, offer(It, P), T)) \\ \alpha_1 = utt(dutch, w_3, inform(Au, auct, A, buyer, offer(It, P), T_2)) \\ \alpha_2 = prh(dutch, w_4, inform(A, buyer, Au, auct, bid(It, P_2), T_3)) \end{cases} \quad \text{(III)}$$

$$\left( \begin{array}{c} X = \{\ \alpha_0 \mid \alpha_1 \wedge \neg(\alpha_2 \wedge T_2 > T_1) \wedge T_0 > T_1\ \} \wedge \\ |X| = 1 \wedge oav(A_1, credit, C) \wedge C < P \end{array} \right) \rightsquigarrow \left( \begin{array}{c} \ominus oav(A_1, credit, C) \wedge \\ \oplus oav(A_1, credit, C_2) \wedge \oplus\alpha_3 \wedge \\ \oplus(C_2 = C - P * 0.1) \wedge \oplus(P_2 = P * 1.2) \end{array} \right)$$

$$where \begin{cases} \alpha_0 = utt(dutch, w_4, inform(A_1, buyer, Au, auct, bid(It, P), T_0)) \\ \alpha_1 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T_1)), \\ \alpha_2 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T_2)) \\ \alpha_3 = obl(dutch, w_5, inform(Au, auct, all, buyer, offer(It, P_2), T_3)) \end{cases} \quad \text{(IV)}$$

$$\left( \begin{array}{c} ctr(dutch, w_5, T_n) \wedge \alpha_0 \wedge \neg(\alpha_1 \wedge T_2 > T) \wedge \\ timeout(dutch, w_4, w_5, T_3) \wedge T_3 > T \wedge \\ oav(IT, reservation\_price, RP) \wedge \\ oav(IT, decrement\_rate, DR) \wedge RP < P - DR \end{array} \right) \rightsquigarrow \left( \begin{array}{c} \oplus\alpha_2 \wedge \\ \oplus(P2 = P - DR) \end{array} \right)$$

$$where \begin{cases} \alpha_0 = utt(dutch, w3, inform(Au, auct, all, buyer, offer(IT, P), T)) \\ \alpha_1 = utt(dutch, w3, inform(Au, auct, all, buyer, offer(IT, P), T_2)) \\ \alpha_2 = obl(dutch, w5, inform(Au, auct, all, buyer, offer(IT, P2), T_4)) \end{cases} \quad \text{(V)}$$

$$\left( \begin{array}{c} ctr(dutch, w_5, T_n) \wedge \alpha_0 \wedge \neg(\alpha_1 \wedge T_2 > T) \wedge \\ timeout(dutch, w_4, w_5, T_3) \wedge T_3 > T \wedge oav(It, reservation\_price, RP) \wedge \\ oav(It, decrement\_rate, DR) \wedge RP \geq P - DR \end{array} \right) \rightsquigarrow \left( \oplus\alpha_2 \right)$$

$$where \begin{cases} \alpha_0 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T)) \\ \alpha_1 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T_2)) \\ \alpha_2 = obl(dutch, w_5, inform(Au, auct, all, buyer, withdrawn(It), T_3)) \end{cases} \quad \text{(VI)}$$

Fig. 2. Rules for the Dutch Auction Protocol

to the price of the good in auction, the rule adds the obligation for the auctioneer to inform all the buyers about the sale. **III. Prevention** – It prevents agents from issuing bids they cannot afford (their credit is insufficient). It states that if agent $Ag$'s credit is less than $P$ (the last offer the auctioneer called for item $It$, at state $w_3$ of scene $dutch$), then agent $Ag$ is prohibited to bid. **IV. Punishment** – It punishes agents when issuing a winning bid they cannot pay for. More precisely, the rule punishes an agent $A_1$ by decreasing his credit of 10% of the value of the good being auctioned. The $oav$ predicate on the *LHS* of the rule represents the current credit of the offending agent. The rule also adds an obligation for the auctioneer to restart the bidding round and the constraint that the new offer should be greater than 120% of the old price. **V. No bids/New Price** – It checks if there were no bids and the next price is greater than the reservation price. If so, it adds the obligation for the auctioneer to start a new bidding round. Rule 5 checks that the current scene state is $w_5$, whether a timeout has expired after the last offer and whether the new price is greater than reservation price. If so, the rule adds the obligation for the auctioneer to offer the item at a lower price. By retrieving the last offer we gather the last offer price. By checking the $oav$ predicates we gather the values of the reservation price and the decrement rate for item $It$.

**VI. No bids/withdrawal** – It checks if there were no bids and the next price is less than the reservation price, then adds the obligation for the auctioneer to withdraw the item. Rule 6 checks that the current state is $w_5$, whether a timeout has occurred after the last offer and whether the new offer price is greater than reservation price. If the *LHS* holds, the rule fires to add the obligation for the auctioneer to withdraw the item. By checking the last offer we gather the last offer price. By checking the $oav$ predicates we gather the values of the reservation price and the decrement rate for the price of item $It$.

This example illustrate how our language addresses the pragmatic concerns raised in section 2.

## 6. CONCLUSIONS AND FUTURE WORK

In this paper we have introduced a formalism for the explicit management of the normative position of agents in electronic institutions. Ours is a rule language in which constraints can be specified and changed at run-time, conferring expressiveness and precision on our constructs. The semantics of our formalism defines a kind of production system in which rules are exhaustively applied to a state of affairs, leading to the next state of affairs. The normative positions are updated via rules, depending on the messages agents send. Our formalism addresses the points of a desiderata for normative languages introduced in section 2. We have explored our proposal in this paper by specifying a version of the Dutch Auction protocol.

We would like to generalise our language to cope with arbitrary actions, rather than just speech acts among agents – this would allow our work to address any type of open multi-agent system. We would also like to improve the semantics of the language in order to support the use of temporal operators for the management of time.

## REFERENCES


ARTIKIS, A., KAMARA, L., PITT, J., AND SERGOT, M. 2005. A Protocol for Resource Sharing in Norm-Governed Ad Hoc Networks. LNCS, vol. 3476. Springer-Verlag.

AXELROD, R. 1997. *The complexity of cooperation: agent-based models of competition and collaboration*. Princeton studies in complexity. Princeton University, New Jersey.

DIGNUM, F. 1999. Autonomous Agents with Norms. *Artificial Intelligence and Law 7*, 1, 69–79.

ESTEVA, M. 2003. Electronic Institutions: from Specification to Development. Ph.D. thesis, Univ. Politècnica de Catalunya (UPC). IIIA monography Vol. 19.

FITTING, M. 1990. *First-Order Logic and Automated Theorem Proving*. Springer-Verlag, New York, U.S.A.

GARCÍA-CAMINO, A., NORIEGA, P., AND RODRÍGUEZ-AGUILAR, J. A. 2005a. Implementing Norms in Electronic Institutions. In *Procs. 4th AAMAS*.

GARCÍA-CAMINO, A., RODRÍGUEZ-AGUILAR, J. A., SIERRA, C., AND VASCONCELOS, W. 2005b. A Distributed Architecture for Norm-Aware Agent Societies. AUCS/TR0503, Dept of Computing Sc., Univ. of Aberdeen, Aberdeen, UK.

JENNINGS, N., SYCARA, K., AND WOOLDRIDGE, M. 1998. A roadmap of agent research and development. *Journal of Agents and Multi-Agents Systems 1*, 7–38.

LÓPEZ Y LÓPEZ, F. 2003. Social power and norms: Impact on agent behaviour. Ph.D. thesis, Univ. of Southampton.

NORIEGA, P. 1997. Agent-Mediated Auctions: The Fishmarket Metaphor. Ph.D. thesis, Universitat Autònoma de Barcelona (UAB). IIIA monography Vol. 8.

SEARLE, J. 1969. *Speech Acts, An Essay in the Philosophy of Language*. Cambridge University Press.

SERGOT, M. 2001. A Computational Theory of Normative Positions. *ACM Trans. Comput. Logic 2*, 4, 581–622.

SHOHAM, Y. AND TENNENHOLTZ, M. 1995. On Social Laws for Artificial Agent Societies: Off-line Design. *Artificial Intelligence 73*, 1-2, 231–252.

WOOLDRIDGE, M. 2002. *An Introduction to Multiagent Systems*. John Wiley & Sons, Chichester, UK.