# Normative Multi-Agent Programs and Their Logics

Mehdi Dastani[1], Davide Grossi[2], John-Jules Ch. Meyer[1], and Nick Tinnemeier[1]

[1] Universiteit Utrecht
The Netherlands
[2] Computer Science and Communication
University of Luxembourg, Luxembourg

**Abstract.** Multi-agent systems are viewed as consisting of individual agents whose behaviors are regulated by an organization artefact. This paper presents a simplified version of a programming language that is designed to implement norm-based artefacts. Such artefacts are specified in terms of norms being enforced by monitoring, regimenting and sanctioning mechanisms. The syntax and operational semantics of the programming language are introduced and discussed. A logic is presented that can be used to specify and verify properties of programs developed in this language.

## 1 Introduction

In this paper, multi-agent systems are considered as consisting of individual agents that are autonomous and heterogenous. Autonomy implies that each individual agent pursues its own objectives and heterogeneity implies that the internal states and operations of individual agents may not be known to external entities [14, 7]. In order to achieve the overall objectives of such multi-agent systems, the observable/external behavior of individual agents and their interactions should be regulated/coordinated.

There are two main approaches to regulate the external behavior of individual agents. The first approach is based on coordination artefacts that are specified in terms of low-level coordination concepts such as synchronization of processes[12]. The second approach is motivated by organizational models, normative systems, and electronic institutions[13, 10, 7, 8]. In such an approach, norm-based artefacts are used to regulate the behavior of individual agents in terms of norms being enforced by monitoring, regimenting and sanctioning mechanisms. Generally speaking, the social and normative perspective is conceived as a way to make the development and maintenance of multi-agent systems easier to manage. A plethora of social concepts (e.g., roles, social structures, organizations, institutions, norms) has been introduced in multi-agent system methodologies (e.g. Gaia [14]), models (e.g. OperA [6], $\mathcal{M}$oise$^+$ [9], electronic institutions and frameworks (e.g. AMELI [7], $\mathcal{S}$-$\mathcal{M}$oise$^+$ [9]).

The main contribution of this paper is twofold. On the one hand, a simplified version of a programming language is presented that is designed to implement

Dagstuhl Seminar Proceedings 08361
Programming Multi-Agent Systems
http://drops.dagstuhl.de/opus/volltexte/2008/1637

multi-agent systems in which the observable (external) behavior of individual agents is regulated by means of norm-based artefacts. Such artefacts are implemented in terms of social concepts such as norms and sanctions, monitor the actions performed by individual agents, evaluate their effects, and impose sanctions if necessary. On the other hand, we devise a logic to specify and verify properties of programs that implement norm-based artefacts.

In order to illustrate the idea of norm-based artefacts, consider the following simple example of a simulated train station where agents ought to buy a ticket before entering the platform or trains. To avoid the queue formation, agents are not checked individually before allowing them to enter the platform or trains. In this simulation, being on the platform without a ticket is considered as a violation and getting on the train without having a ticket is considered as a more severe violation. A norm-based artefact detects (all or some) violations by (all or some) agents and reacts on them by issuing a fine if the first violation occurs, for instance by charging the credit card of the defecting user, and a higher fine if the second violation occurs.

In this paper, we first briefly explain our idea of normative multi-agent systems and discuss two norm-based approaches to multi-agent systems, that is, ISLANDER/AMELI [7] and S-MOISE+ [9]. In section 3, we present the syntax and operational semantics of a programming language designed to implement normative multi-agent systems. This programming language allows the implementation of norm-based artefacts by providing programming constructs to represent norms and mechanisms to enforce them. In section 4, a logic is presented that can be used to specify and verify properties of norm-based artefacts implemented in the presented programming language. Finally, in section 5, we conclude the paper and discuss some future directions in this research area.

## 2 Norms and Multi-Agent Systems

Norms in multi-agent systems can be used to specify the standards of behavior that agents ought to follow to meet the overall objectives of the system. However, to develop a multi-agent system does not boil down to state a number of standards of behavior in the form of a set of norms, but rather to organize the system in such a way that those standards of behavior are actually followed by the agents. This can be achieved by regimentation [10] or enforcement mechanisms, e.g., [8].

When regimenting norms all agents' external actions leading to a violation of those norms are made impossible. Via regimentation (e.g., gates in train stations) the system prevents an agent from performing a forbidden action (e.g., entering a train platform without a ticket). However, regimentation drastically decreases agent autonomy. Instead, enforcement is based on the idea of responding after a violation of the norms has occurred. Such a response, which includes sanctions, aims to return the system to an acceptable/optimal state. Crucial for enforcement is that the actions that violate norms are observable by the system (e.g., fines can be issued only if the system can detect travelers entering the platform or trains without a ticket). Another advantage of having enforcement over regimentation is that allowing for violations contributes to the flexibility and autonomy of the agent's behavior [3]. These norms are often specified by means of concepts like permissions, obligations, and prohibitions.

In the literature of multi-agent systems related work can be found on electronic institutions. In particular, ISLANDER[7] is a formal framework for specifying norms in institutions, which is used in the AMELI platform [7] for executing electronic institutions based on norms provided in it. However, the key aspect of ISLANDER/AMELI is that norms can never be violated by agents. In other words, systems programmed via ISLANDER/AMELI make only use of regimentation in order to guarantee the norms to be actually followed. This is an aspect which our approach intends to relax guaranteeing higher autonomy to the agents, and higher flexibility to the system.

A work that is concerned with programming multiagent systems using (among others) normative concepts is also S-MOISE+, which is an organizational middleware that follows the Moise+ model[9]. This approach, like ours, builds on programming constructs investigated in social and organizational sciences. However, S-MOISE+ lacks formal operational semantics, which is instead the main contribution of the present paper to the development of programming languages form multi-agent systems. Besides, norms in S-MOISE+ typically lack monitoring and sanctioning mechanisms for their implementation which are, instead, the focus of our proposal. It should be noted that [11] advocates the use of artifacts to implement norm enforcement mechanisms. However, it is not explained how this can be done using those artifacts.

To summarize, ISLANDER/AMELI implements norm via full regimentation, while in S-MOISE+ violations are possible, although no specific system's response to violations is built in the framework. We deem these shortcomings to have a common root, namely the absence of a computational model of norms endowed with a suitable operational semantics. The present paper fills this gap along the same lines that have been followed for the operationalization of BDI notions in the APL-like agent programming languages [5, 4]. Finally, it should be noted that besides normative concepts MOISE+ and ISLANDER/AMELI also provide a variety of other social and organizational concepts. Since the focus of this paper is on the normative aspect, the above discussion is limited hereto. Future research will focus on other social and organizational concepts.

## 3 Programming Multi-Agent Systems with Norms

In this section, we present a programming language to facilitate the implementation of multi-agent systems with norms, i.e., to facilitate the implementation of norm-based artefacts that coordinate/regulate the behavior of participating individual agents. A normative multi-agent system (i.e., a norm-based artefact) is considered to contain two modules: an organization module that specifies norms and sanctions, and an environment module in which individual agents can perform actions. The individual agents are assumed to be implemented in a programming language, not necessarily known to the multi-agent system programmer, though the programmer is required to have the reference to the (executable) programs of each individual agent. It is also assumed that all actions that are performed by individual agents are observable to the multi-agent system (i.e., norm-based artefact). Note that the reference to the (executable) programs of individual agents are required such that multi-agent systems (i.e., normative artefact) can observe the actions generated by the agent programs. Finally, we assume that the effect of an individual agent's action in the external environment is determined by the program that implements the norm-based artefact (i.e., by the multi-agent system program). Most noticeably it is not assumed that the agents are able to reason about the norms of the system.

The programming language for normative multi-agent systems provides programming constructs to specify the effect of an agent's actions in the environment, norms, sanctions, and the initial state of the environment. Moreover, the programming language is based on a monitoring and a sanctioning mechanism that observes the actions performed by the agents, determines their effects in the shared environment, determines the violations caused by performing the actions, and possibly, imposes sanctions. A program in this language is the implementation of a norm-based artefact. As we assume that the norm-based artefacts determine the effects of external actions in the shared environment, the programming language should provide constructs to implement these effects. The effect of an agent's (external) actions is specified by a set of literals that should hold in the shared environment after the external action is performed by the agent. As external actions can have different effects when they are executed in different states of the shared environment, we add a set of literals that function as the pre-condition of those effect.

We consider norms as being represented by counts-as rules [13], which ascribe "institutional facts" (e.g. "a violation has occurred"), to "brute facts" (e.g. "an agent is on the train without ticket"). For example, a counts-as rule may express the norm "an agent on the train without ticket counts-as a violation". In our framework, brute facts constitute the environment shared by the agents, while institutional facts constitute the normative/institutional state of the multi-agent system. Institutional facts are used with the explicit aim of triggering system's reactions (e.g., sanctions). As showed in [8] counts-as rules can enjoy a rather classical logical behavior, and are here implemented as simple rules that relate brute and normative facts. In the presented programming language, we distinguish brute facts from normative (institutional) facts and assume two disjoint sets of propositions to denote these facts.

Brute and institutional facts constitute the (initial) state of the multi-agent system (i.e., the state of the norm-based artefact). Brute facts are initially set by the programmer by means of the initial state of the shared environment. These facts can change as individual agents perform actions in the shared environment. Normative facts are determined by applying counts-as rules in multi-agent states. The application of counts-as rules in subsequent states of a multi-agent system realizes a monitoring mechanism as it determines and detects norm violations during the execution of the multi-agent system.

Sanctions are also implemented as rules, but follow the opposite direction of counts-as rules. A sanction rule determines which brute facts will be brought about by the system as a consequence of the normative facts. Typically, such brute facts are sanctions, such as fines. Notice that in human systems sanctions are usually issued by specific agents (e.g. police agents). This is not the case in our computational setting, where sanctions necessarily follow the occurrence of a violation if the relevant sanction rule is in place (comparable to automatic traffic control and issuing tickets). It is important to stress, however, that this is not an intrinsic limitation of our approach. We do not aim at mimicking human institutions but rather providing the specification of computational systems.

## 3.1 Syntax.

In order to represent brute and institutional facts in our normative multi-agent systems programming language, we introduce two disjoint sets of propositions to denote these facts. The syntax of the normative multi-agent system programming language is presented below using the EBNF notation. In the following, we use `<b-prop>` and `<i-prop>` to be propositional formulae taken from two different disjoint sets of propositions. Moreover, we use `<ident>` to denote a string and `<int>` to denote an integer.

```
N-MAS_Prog   := "Agents: " (<agentName> <agentProg> [<nr>])+ ;
                "Facts: " <bruteFacts>
                "Effects: " <effects>
                "Counts-as rules: " <counts-as>
                "Sanction rules: " <sanctions>;
<agentName>  := <ident>;
<agentProg>  := <ident>;
<nr>         := <int>;
<bruteFacts> := <b-literals>;
<effects>    := ({<b-literals>} <actionName> {<b-literals>})+;
<counts-as>  := ( <literals> ⇒ <i-literals> )+;
<sanctions>  := ( <i-literals> ⇒ <b-literals>)+;
<actionName> := <ident>;
<b-literals> := <b-literal> {"," <b-literal>};
<i-literals> := <i-literal> {"," <i-literal>};
<literals>   := <literal> {"," <literal>};
<literal>    := <b-literal> | <i-literal>;
<b-literal>  := <b-prop> | "not" <b-prop>;
<i-literal>  := <i-prop> | "not" <i-prop>;
```

In order to illustrate the use of this programming language, consider the following underground station example.

```
Agents:          passenger   PassProg   1
Facts:           {-at_platform, -in_train, -ticket}
Effects:         {-at_platform} enter {at_platform},
                 {-ticket} buy_ticket {ticket},
                 {at_platform, -in_train} embark {-at_platform, in_train}
Counts_as rules: {at_platform , -ticket} ⇒ {viol_1},
                 {in_train , -ticket} ⇒ {viol_⊥}
Sanction rules:  {viol_1} ⇒ {fined_10}
```

This program creates one agent called passenger whose (executable) specification is included in a file with the name PassProg. The Facts, which implement brute facts, determine the initial state of the shared environment. In this case, the agent is not at the platform (-at_platform) nor in the train (-in_train) and has no ticket (-ticket). The Effects indicate how the environment can advance in its computation. Each effect is of the form {pre-condition} action {post-condition}. The first effect, for instance, means that if the agent performs an enter action when not at the platform, the result is that the agent is on the platform (either with or without a ticket). Only those effects that are changed are thus listed in the post-condition. The Counts_as rules determine the normative effects for a given (brute and normative) state of the multi-agent system. The first rule, for example, states that being on the platform without having a ticket is a specific violation (marked by $\text{viol}_1$). The second rule marks states where agents are on a train without a ticket with the specifically designated literal $\text{viol}_\perp$. This literal is used to implement regimentation. The operational semantics of the language ensures that the designated literal $\text{viol}_\perp$ can never hold during any run of the system (see Definition 3). Intuitively, rules with $\text{viol}_\perp$ as consequence could be thought of as placing gates blocking an agent's action. Finally, the aim of Sanction rules is to determine the punishments that are imposed as a consequence of violations. In the example the violation of type $\text{viol}_1$ causes the sanction $\text{fined}_{10}$ (e.g., a 10 EUR fine).

Counts-as rules obey syntactic constraints. Let $l = (\Phi \Rightarrow \Psi)$ be a rule, we use $\texttt{cond}_l$ and $\texttt{cons}_l$ to indicate the condition $\Phi$ and consequent $\Psi$ of the rule $l$, respectively. We consider only sets of rules such that 1) they are finite; 2) they are such that each condition has exactly one associated consequence (i.e., all the consequences of a given conditions are packed in one single set cons); and 3) they are such that for counts-as rule $k, l$, if $\texttt{cons}_k \cup \texttt{cons}_l$ is inconsistent (i.e., contains $p$ and $\neg p$), then $\texttt{cond}_k \cup \texttt{cond}_l$ is also inconsistent. That is to say, rules trigger inconsistent conclusions only in different states. In the rest of this paper, sets of rules enjoying these three properties are denoted by **R**.

## 3.2 Operational Semantics.

One way to define the semantics of this programming language is by means of operational semantics. Using such semantics, one needs to define the configuration (i.e., state) of normative multi-agent systems and the transitions that such configurations can undergo through transition rules. The state of a multi-agent system with norms consists of the state of the external environment, the normative state, and the states of individual agents.

**Definition 1.** *(Normative Multi-Agent System Configuration) Let $P_b$ and $P_n$ be two disjoint sets of literals denoting atomic brute and normative facts (including* $\mathrm{viol}_\perp$*), respectively. Let $A_i$ be the configuration of individual agent $i$. The configuration of a normative multi-agent system is defined as $\langle \mathcal{A}, \sigma_b, \sigma_n \rangle$ where $\mathcal{A} = \{A_1, \ldots, A_n\}$, $\sigma_b$ is a consistent set of literals from $P_b$ denoting the brute state of multi-agent system and $\sigma_n$ is a consistent set of literals from $P_n$ denoting the normative state of multi-agent system.*

The configuration of such a multi-agent system can change for various reasons, e.g., because individual agents perform actions in the external environment or because the external environment can have its own internal dynamics (the state of a clock changes independent of an individual agent's action). In operational semantics, transition rules specify how and when configurations can change, i.e., they specify which transition between configurations are allowed and when they can be derived. In this paper, we consider only the transition rules that specify the transition of multi-agent system configurations as a result of performing external actions by individual agents. Of course, individual agents can perform (internal) actions that modify only their own configurations and have no influence on the multi-agent system configuration. The transition rules to derive such transitions are out of the scope of this paper.

**Definition 2.** *(Transitions of Individual Agent's Actions) Let $A_i$ and $A'_i$ be configurations of individual agent $i$, and $\alpha(i)$ be an (observable) external action performed by agent $i$. Then, the following transition captures the execution of an external action by an agent.*

$$A_i \xrightarrow{\alpha(i)} A'_i \ : \textit{agent } i \textit{ can perform external action } \alpha$$

This transition indicates that an agent configuration can change by performing an external action. The performance of the external action is broadcasted to the multi-agent system level. Note that no assumption is made about the internals of individual agents as we do not present transition rules for deriving internal agent transitions (denoted as $A \longrightarrow A'$). The only assumption is that the action of the agent is observable. This is done by labeling the transition with the external action name.

Before presenting the transition rule specifying the possible transitions of the normative MAS configurations, the closure of a set of conditions under a set of (counts-as and sanction) rules needs to be defined. Given a set $\mathbf{R}$ of rules and a set $X$ of literals, we define the set of applicable rules in $X$ as $\mathtt{Appl}^{\mathbf{R}}(X) = \{\Phi \Rightarrow \Psi \mid X \models \Phi\}$. The closure of $X$ under $\mathbf{R}$, denoted as $\mathtt{Cl}^{\mathbf{R}}(X)$, is inductively defined as follows:

**B:** $\mathtt{Cl}^{\mathbf{R}}_0(X) = X \cup (\bigcup_{l \in \mathtt{Appl}^{\mathbf{R}}(X)} \mathtt{cons}_l)$
**S:** $\mathtt{Cl}^{\mathbf{R}}_{n+1}(X) = \mathtt{Cl}^{\mathbf{R}}_n(X) \cup (\bigcup_{l \in \mathtt{Appl}^{\mathbf{R}}(\mathtt{Cl}^{\mathbf{R}}_n(X))} \mathtt{cons}_l)$

Because of the properties of finiteness, consequence uniqueness and consistency of $\mathbf{R}$ one and only one finite number $m+1$ can always be found such that $\mathtt{Cl}^{\mathbf{R}}_{m+1}(X) = \mathtt{Cl}^{\mathbf{R}}_{m}(X)$ and $\mathtt{Cl}^{\mathbf{R}}_{m}(X) \neq \mathtt{Cl}^{\mathbf{R}}_{m-1}(X)$. Let such $m+1$ define the closure $X$ under $\mathbf{R}$: $\mathtt{Cl}^{\mathbf{R}}(X) = \mathtt{Cl}^{\mathbf{R}}_{m+1}(X)$. Note that the closure may become inconsistent due to the ill-defined set of counts-as rules. For example, the counts-as rule $p \Rightarrow -p$ (or the set of counts as rules $\{p \Rightarrow q\ ,\ q \Rightarrow -p\}$), where $p$ and $q$ are normative facts, may cause the normative state of a multi-agent system to become inconsistent.

We can now define a transition rule to derive transitions between normative multi-agent system configurations. In this transition rule, the function *up* determines the effect of action $\alpha(i)$ on the environment $\sigma_b$ based on its specification $(\Phi\ \alpha(i)\ \Phi')$ as follows:

$$up(\alpha(i), \sigma_b) = (\sigma_b \cup \Phi') \setminus (\{p \mid -p \in \Phi'\} \cup \{-p \mid p \in \Phi'\})$$

**Definition 3.** *(Transition Rule for Normative Multi-Agent Systems) Let* $\mathbf{R_c}$ *be the set of counts-as rules,* $\mathbf{R_s}$ *be the set of sanction rules, and* $(\Phi\ \alpha(i)\ \Phi')$ *be the specification of action* $\alpha(i)$*. The multi-agent transition rule for the derivation of normative multi-agent system transitions is defined as follows:*

$$\frac{\begin{array}{c} A_i \in \mathcal{A} \quad \& \quad A_i \overset{\alpha(i)}{\rightarrow} A'_i \quad \& \quad \sigma_b \models \Phi \quad \& \quad \sigma'_b = up(\alpha(i), \sigma_b) \\ \sigma'_n = \mathtt{Cl}^{\mathbf{R_c}}(\sigma'_b) \setminus \sigma'_b \quad \& \quad \sigma'_n \not\models \mathrm{viol}_\perp \quad \& \quad S = \mathtt{Cl}^{\mathbf{R_s}}(\sigma'_n) \setminus \sigma'_n \quad \& \quad \sigma'_b \cup S \not\models \perp \end{array}}{\langle \mathcal{A}, \sigma_b, \sigma_n \rangle \longrightarrow \langle \mathcal{A}', \sigma'_b \cup S, \sigma'_n \rangle}$$

*where* $\mathcal{A}' = (\mathcal{A} \setminus \{A_i\}) \cup \{A'_i\}$ *and* $\mathrm{viol}_\perp$ *is the designated literal for regimentation.*

This transition rule captures the effects of performing an external action by an individual agent on both external environments and the normative state of the MAS. First, the effect of $\alpha$ on $\sigma_b$ is computed. Then, the updated environment is used to determine the new normative state of the system by applying all counts-as rules to the new state of the external environments. Finally, possible sanctions are added to the new environment state by applying sanction rules to the new normative state of the system. In should be emphasized that other multi-agent transition rules, such as transition rules for communication actions, are not presented in this paper because the focus here is on how norms determine the effects of external actions.

Note that the external action of an agent can be executed only if it would not result in a state containing $\mathrm{viol}_\perp$. This captures exactly the regimentation of norms. Hence, once assumed that the initial normative state does not include $\mathrm{viol}_\perp$, it is easy to see that the system will never be in a $\mathrm{viol}_\perp$-state. It is important to note that when a normative state $\sigma'_n$ becomes inconsistent, the proposed transition rule cannot be applied because an inconsistent $\sigma'_n$ entails $viol_\perp$. Also, note that the condition $\sigma'_b \cup S \not\models \perp$ guarantees that the environment state never can become inconsistent. Finally, it should be emphasized that the normative state $\sigma'_b$ is not defined on $\sigma_n$ and is always computed anew.

## 4 Logic

In this section, we propose a logic to specify and verify liveness and safety properties of multi-agent system programs with norms. This logic, which is a variant of Propositional Dynamic Logic (PDL, see [2]), is in the spirit of [1] and rely on that work. It is important to note that the logic developed in [1] aims at specifying and verifying properties of single agents programmed in terms of beliefs, goals, and plans. Here we modify the logic and apply it to multi-agent system programs. We first introduce some preliminaries before presenting the logic.

### 4.1 Preliminaries

We show how the programming constructs can be used for grounding a logical semantics. Let $P$ denote the set of propositional variables used to describe brute and normative states of the system. It is assumed that each propositional variable in $P$ denotes either an institutional/normative or a brute state-of-affairs: $P = P_n \cup P_b$ and $P_n \cap P_b = \emptyset$. A state $s$ is represented as a pair $\langle \sigma_b, \sigma_n \rangle$ where $\sigma_b = \{(-)p_1, \dots, (-)p_n : p_i \in P_b\}$ is a consistent set of literals (i.e., for no $p \in P_b$ it is the case that $p \in \sigma_b$ and $-p \in \sigma_b$), and $\sigma_n$ is like $\sigma_b$ for $P_n$.

Rules are pairs of conditions and consequences $(\{(-)p_1, \dots, (-)p_n \mid (-)p_i \in X\}, \{(-)q_1, \dots, (-)q_k \mid (-)q_i \in Y\})$ with $X$ and $Y$ being either $\sigma_b$ or $\sigma_n$ when applied in state $\langle \sigma_b, \sigma_n \rangle$. Following [8], if $X = \sigma_b$ and $Y = \sigma_n$ then the rule is called *bridge counts-as rule*; if $X = Y = \sigma_n$ then the rule is an *institutional counts-as rule*; if $X = \sigma_n$ and $Y = \sigma_b$ then the rule is a *sanction rule*. Literals $p$'s and $q$'s are taken to be disjoint. Leaving technicalities aside, bridge counts-as rules connect brute states to normative/institutional ones, institutional counts-as rules connect institutional facts to institutional facts, and sanction rules connect normative states to brute ones.

Given a set $\mathbf{R}$ of rules, we say a state $s = \langle \sigma_b, \sigma_n \rangle$ to be $\mathbf{R}$-aligned if for all pairs $(\mathtt{cond}_k, \mathtt{cons}_k)$ in $\mathbf{R}$: if $\mathtt{cond}_k$ is satisfied either by $\sigma_b$ (in the case of a bridge counts-as rule) or by $\sigma_n$ (in the case of an institutional counts-as or a sanction rule), then $\mathtt{cons}_k$ is satisfied by $\sigma_n$ (in the case of a bridge or institutional counts-as rule) or by $\sigma_b$ (in the case of a sanction rule), respectively. States that are $\mathbf{R}$-aligned are states which instantiate the normative system specified by $\mathbf{R}$.

Let the set of agents' external actions $\mathtt{Ac}$ be the union $\bigcup_{i \in I} \mathtt{Ac}_i$ of the finite sets $\mathtt{Ac}_i$ of external actions of each agent $i$ in the set $I$. We denote external actions as $\alpha(i)$ where $\alpha \in \mathtt{Ac}_i$ and $i \in I$. We associate now with each $\alpha(i) \in \mathtt{Ac}_i$ a set of pre- and post-conditions $\{(-)p_1 \in \sigma_b, \dots, (-)p_n \in \sigma_b\}$, $\{(-)q_1 \in \sigma'_b, \dots, (-)q_k \in \sigma'_b\}$ (where $p$'s and $q$'s are not necessarily disjoint) when $\alpha(i)$ is executed in a state with brute facts set $\sigma_b$ which satisfies the pre-condition then the resulting state $s'$ has the brute facts set $\sigma'_b$ which satisfies the post-condition (including replacing $p$ with $-p$ if necessary to preserve consistency) and it is such that *the rest of* $\sigma'_b$ *is the same as* $\sigma_b$. Executing an action $\alpha(i)$ in different configurations may give different results. For each $\alpha(i)$, we denote the set of pre- and post-condition pairs $\{(\mathtt{prec}_1, \mathtt{post}_1), \dots, (\mathtt{prec}_m, \mathtt{post}_m)\}$ by $C_b(\alpha(i))$. We assume that $C_b(\alpha(i))$ is finite, that pre-conditions $\mathtt{prec}_k, \mathtt{prec}_l$ are mutually exclusive if $k \neq l$, and that each pre-condition has exactly one associated post-condition. We denote the set of all such pre- and post-conditions of all agents' external actions by $\mathbf{C}$.

Now everything is put into place to show how the execution of $\alpha(i)$ in a state with brute facts set $\sigma_b$ also univocally changes the normative facts set $\sigma_n$ by means of the applicable counts-as rules, and adds the resulting sanctions by means of the applicable sanction rules. If $\alpha(i)$ is executed in a state $\langle \sigma_b, \sigma_n \rangle$ with brute facts set $\sigma_b$, which satisfies the pre-conditions, then the resulting state $\langle \sigma'_b \cup S, \sigma'_n \rangle$ is such that $\sigma'_b$ satisfies the brute post-condition of $\alpha(i)$ (including replacing $p$ with $-p$ if necessary) and the rest of $\sigma'_b$ is the same of $\sigma_b$; $\sigma'_n$ is determined by the closure of $\sigma'_b$ with counts-as rules $\mathbf{R}_c$; sanctions $S$ are obtained via closure of $\sigma'_n$ with sanction rules $\mathbf{R}_s$.

## 4.2 Language

The language $L$ for talking about normative multi-agent system programs is just the language of PDL built out of a finite set of propositional variables $P \cup -P$ (i.e., the literals built from $P$), used to describe the system's normative and brute states, and a finite set $\mathtt{Ac}$ of agents' actions. Program expressions $\rho$ are built out of external actions $\alpha(i)$ as usual, and formulae $\phi$ of $L$ are closed under boolean connectives and modal operators:

$$\rho ::= \alpha(i) \mid \rho_1 \cup \rho_2 \mid \rho_1 ; \rho_2 \mid ?\phi \mid \rho^*$$
$$\phi ::= (-)p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \langle \rho \rangle \phi$$

with $\alpha(i) \in \mathtt{Ac}$ and $(-)p \in P \cup -P$. Connectives $\vee$ and $\rightarrow$, and the modal operator $[\rho]$ are defined as usual.

## 4.3 Semantics.

The language introduced above is interpreted on transition systems that generalize the operational semantics presented in the earlier section, in that they do not describe a particular program, but all possible programs —according to $\mathbf{C}$— generating transitions between all the $\mathbf{R_c}$ and $\mathbf{R_s}$-aligned states of the system. As a consequence, the class of transition systems we are about to define will need to be parameterized by the sets $\mathbf{C}$, $\mathbf{R_c}$ and $\mathbf{R_s}$.

A model is a structure $M = \langle S, \{R_{\alpha(i)}\}_{\alpha(i) \in \mathtt{Ac}}, V \rangle$ where:

- $S$ is a set of $\mathbf{R_c}$ and $\mathbf{R_s}$-aligned states.
- $V = (V_b, V_n)$ is the evaluation function consisting of brute and normative valuation functions $V_b$ and $V_n$ such that for $s = \langle \sigma_b, \sigma_n \rangle$, $V_b(s) = \sigma_b$ and $V_n(s) = \sigma_n$.
- $R_{\alpha(i)}$, for each $\alpha(i) \in \mathtt{Ac}$, is a relation on $S$ such that $(s, s') \in R_{\alpha(i)}$ iff for some $(\mathtt{prec}_k, \mathtt{post}_k) \in C(\alpha(i))$, $\mathtt{prec}_k(s)$ and $\mathtt{post}_k(s')$, i.e., for some pair of pre- and post-conditions of $\alpha(i)$, the pre-condition holds for $s$ and the corresponding post-condition holds for $s'$. Note that this implies two things.

First, an $\alpha(i)$ transition can only originate in a state $s$ which satisfies one of the pre-conditions for $\alpha(i)$. Second, since pre-conditions are mutually exclusive, every such $s$ satisfies exactly one pre-condition, and all $\alpha(i)$-successors of $s$ satisfy the matching post-condition.

Given the relations corresponding to agents' external actions in $M$, we can define sets of paths in the model corresponding to any PDL program expression $\rho$ in $M$. A set of paths $\tau(\rho) \subseteq (S \times S)^*$ is defined inductively:

- $\tau(\alpha(i)) = \{(s, s') : R_{\alpha(i)}(s, s')\}$
- $\tau(\phi?) = \{(s, s) : M, s \models \phi\}$
- $\tau(\rho_1 \cup \rho_2) = \{z : z \in \tau(\rho_1) \cup \tau(\rho_2)\}$
- $\tau(\rho_1; \rho_2) = \{z_1 \circ z_2 : z_1 \in \tau(\rho_1),\ z_2 \in \tau(\rho_2)\}$, where $\circ$ is concatenation of paths , such that $z_1 \circ z_2$ is only defined if $z_1$ ends in the state where $z_2$ starts
- $\tau(\rho^*)$ is the set of all paths consisting of zero or finitely many concatenations of paths in $\tau(\rho)$ (same condition on concatenation as above)

Constructs such as `If` $\phi$ `then` $\rho_1$ `else` $\rho_2$ and `while` $\phi$ `do` $\rho$ are defined as $(\phi?; \rho_1) \cup (\neg\phi?; \rho_2)$ and $(\phi?; \rho)^*; \neg\phi$, respectively. The satisfaction relation $\models$ is inductively defined as follows:

- $M, s \models (-)p$ iff $(-)p \in V_b(s)$ for $p \in P_b$
- $M, s \models (-)p$ iff $(-)p \in V_n(s)$ for $p \in P_n$
- $M, s \models \neg\phi$ iff $M, s \not\models \phi$
- $M, s \models \phi \wedge \psi$ iff $M, s \models \phi$ and $M, s \models \psi$
- $M, s \models \langle\rho\rangle\phi$ iff there is a path in $\tau(\rho)$ starting in $s$ which ends in a state $s'$ such that $M, s' \models \phi$.
- $M, s \models [\rho]\phi$ iff for all paths $\tau(\rho)$ starting in $s$, the end state $s'$ of the path satisfies $M, s' \models \phi$.

Let the class of transition systems defined above be denoted $\mathbf{M}_{\mathbf{C},\mathbf{R_c},\mathbf{R_s}}$ where $\mathbf{C}$ is the set of pre- and post-conditions of external actions, $\mathbf{R_c}$ is the set of counts-as rules and $\mathbf{R_s}$ the set of sanction rules.

## 4.4 Axiomatics.

The axiomatics shows in what the logic presented differs w.r.t. standard PDL. In fact, it is a conservative extension of PDL with domain-specific axioms needed to axiomatize the behavior of normative multi-agent system programs.

For every pre- and post-condition pair $(\mathtt{prec}_i, \mathtt{post}_i)$ we describe states satisfying $\mathtt{prec}_i$ and states satisfying $\mathtt{post}_i$ by formulas of $L$. More formally, we define a formula $\mathit{tr}(X)$ corresponding to a pre- or post-condition $X$ as follows: $\mathit{tr}((-)p) = (-)p$ and $\mathit{tr}(\{\phi_1, \ldots, \phi_n\}) = \mathit{tr}(\phi_1) \wedge \ldots \wedge \mathit{tr}(\phi_n)$. This allows us to axiomatize pre- and post-conditions of actions. The conditions and consequences of counts-as rules and sanction rules can be defined in similar way as pre- and post-conditions of actions, respectively. The set of models $\mathbf{M}_{\mathbf{C},\mathbf{R_c},\mathbf{R_s}}$ is axiomatized as follows:

**PDL** Axioms and rules of PDL

**Ax Consistency** Consistency of literals: $\neg(p \wedge -p)$

**Ax Counts-as** For every rule $(\texttt{cond}_k, \texttt{cons}_k)$ in $\mathbf{R_c}$: $tr(\texttt{cond}_k) \rightarrow tr(\texttt{cons}_k)$

**Ax Sanction** For every rule $(\texttt{viol}_k, \texttt{sanc}_k)$ in $\mathbf{R_s}$: $tr(\texttt{viol}_k) \rightarrow tr(\texttt{sanc}_k)$

**Ax Regiment** $\text{viol}_\bot \rightarrow \bot$

**Ax Frame** For every action $\alpha(i)$ and every pair of pre- and post-conditions $(\texttt{prec}_j, \texttt{post}_j)$ in $C(\alpha(i))$ and formula $\Phi$ built out of $P_b$ not containing any propositional variables occurring in $\texttt{post}_j$:

$$tr(\texttt{prec}_j) \wedge \Phi \rightarrow [\alpha(i)](tr(\texttt{post}_\texttt{j}) \wedge \Phi)$$

This is a frame axiom for actions.

**Ax Non-Executability** For every action $\alpha(i)$, where all possible pre-conditions in $C(\alpha(i))$ are $\texttt{prec}_1, \ldots, \texttt{prec}_k$: $\neg tr(\texttt{prec}_1) \wedge \ldots \wedge \neg tr(\texttt{prec}_k) \rightarrow \neg\langle\alpha(i)\rangle\top$ where $\top$ is a tautology.

**Ax Executability** For every action $\alpha(i)$ and every pre-condition $\texttt{prec}_j$ in $C(\alpha(i))$: $tr(\texttt{prec}_j) \rightarrow \langle\alpha(i)\rangle\top$

Let us call the axiom system above $\mathbf{Ax}_{\mathbf{C},\mathbf{R_c},\mathbf{R_s}}$, where $\mathbf{C}$ is the set of brute pre- and post-conditions of atomic actions, $\mathbf{R_c}$ is the set of counts-as rules, and $\mathbf{R_s}$ is the set of sanction rules.

**Theorem 1.** *Axiomatics* $\mathbf{Ax}_{\mathbf{C},\mathbf{R_c},\mathbf{R_s}}$ *is sound and weakly complete for the class of models* $\mathbf{M}_{\mathbf{C},\mathbf{R_c},\mathbf{R_s}}$.

*Proof.* Soundness is proven as usual by induction on the length of derivations. We sketch the proof of completeness. It builds on the usual completeness proof of PDL via finite canonical models. Given a consistent formula $\phi$ to be proven satisfiable, such models are obtained via the Fischer-Ladner closure of the set of subformulae of the formula $\phi$ extended with all pre- and post-conditions of any action $\alpha(i)$ occurring in $\phi$. Let $FLC(\phi)$ denote such closure. The canonical model consists of all the maximal $\mathbf{Ax}_{\mathbf{C},\mathbf{R_c},\mathbf{R_s}}$-consistent subsets of $FLC(\phi)$. The accessibility relation and the valuation of the canonical model are defined like in PDL and the truth lemma follows in the standard way. It remains to be proven that the model satisfies the axioms. First, since the states in the model are maximal and consistent w.r.t. *Ax Counts-as*, *Ax Sanction*, *Ax Consistency*, and *AxRegiment*, they are $\mathbf{R_c}$- and $\mathbf{R_s}$-aligned, $\sigma_b$ and $\sigma_n$ are consistent, and no state is such that $\sigma_n \models \texttt{viol}_\bot$. Second, it should be shown that the canonical model satisfies the pre- and post-conditions of the actions occurring in $\phi$ in that: a) no action $\alpha(i)$ is executable in a state $s$ if none of its preconditions are satisfied by $s$, and b) if they hold in $s$ then the corresponding post-conditions hold in $s'$ which is accessible by$R_{\alpha(i)}$ from $s$. As to a), if a state $s$ in the canonical model does not satisfy any of the preconditions of $\alpha(i)$ then, by *Ax Non-Executability* and the definition of the canonical accessibility relation, there is no $s'$ in the model such that $sR_{\alpha(i)}s'$. As to b), if a state $s$ in the canonical model satisfies one of the preconditions $\texttt{prec}_\texttt{j}$ of $\alpha(i)$ then$tr(\texttt{prec}_\texttt{j})$ belongs to $s$ and, by *Ax Frame*, $[\alpha(i)]tr(\texttt{post}_\texttt{j})$ also do. Now, *Ax Executability* guarantees that there exists at least one $s'$ such that $sR_{\alpha(i)}s'$, and, for any $s'$ such that $sR_{\alpha(i)}s'$, by the definition of such canonical accessibility relation, $s'$ contains $tr(\texttt{post}_\texttt{j})$ (otherwise it would not be the case that $sR_{\alpha(i)}s'$). On the other hand, for any literal $(-)p$ in $s$ not occurring in $\mathit{tr}(\texttt{post}_\texttt{j})$, its value cannot change from $s$ to $s'$ since, if it would, then for *Ax Frame* it would not be the case that $sR_{\alpha(i)}s'$, which is impossible. This concludes the proof.

## 4.5 Verification

To verify a normative multi-agent system program means, in our perspective, to check whether the program implementing the normative artefact is soundly designed w.r.t. the regimentation and sanctioning mechanisms it is supposed to realize or, to put it in more general terms, to check whether certain property holds in all (or some) states reachable by the execution traces of the multi-agent system program. In order to do this, we need to translate a multi-agent system program into a PDL program expression.

As explained in earlier sections, a multi-agent system program assumes a set of behaviors $A_1, \ldots, A_n$ of agents $1, \ldots, n$, each of which is a sequence of external actions (the agents actions observed from the multi-agent level), i.e., $A_i = \alpha_i^1; \alpha_i^2, \ldots$ where $\alpha_i^j \in Ac$. [1] Moreover, a multi-agent system program with norms consists of an initial set of brute facts, a set of counts-as rules and a set of sanction rules which together determine the initial state of the program. In this paper, we consider the execution of a multi-agent program as interleaved executions of the involved agents' behaviors started at the initial state.

Given $I$ as the set of agents' names and $A_i$ as the behavior of agent $i \in I$, the execution of a multi-agent program can be described as PDL expression $\bigcup\ \mathit{interleaved}(\{A_i | i \ \in I\})$, where $\mathit{interleaved}(\{A_i | i \ \in I\})$ yields all possible interleavings of agents' behaviors, i.e., all possible interleavings of actions from sequences $A_i$. It is important to notice that $\bigcup\ \mathit{interleaved}(\{A_i | i \ \in I\})$ corresponds to the set of computations sequences (execution traces) generated by the operational semantics.

The general verification problem can now be formulated as follows. Given a multi-agent system program with norms in a given initial state satisfying $\phi \in L$, the state reached after the execution of the program satisfies $\psi$, i.e.:

$$\phi \rightarrow \langle [\bigcup\ \mathit{interleaved}(\{A_i | i \ \in I\})] \rangle \psi$$

In the above formulation, the modality $\langle [\ldots] \rangle$ is used to present both safety $[\ldots]$ and liveness $\langle \ldots \rangle$ properties. We briefly sketch a sample of such properties using again the multi-agent system program with norms which implements the train station example with one passenger agent (see Section 3).

**Sanction follows violation.** Entering without a ticket results in a fine, i.e.,

$$\neg\texttt{at\_platform} \wedge \neg\texttt{train} \wedge \neg\texttt{ticket} \rightarrow [\texttt{enter}](\texttt{viol}_1 \wedge \texttt{pay}_{10}).$$

[1] Note an agent's behavior can always be written as a (set of) sequence(s) of actions, which in turn can be written as a PDL expressions.

**Norm obedience avoids sanction.** Buying a ticket if you have none and entering the platform does not result in a fine, i.e.:

$$\neg\texttt{at\_platform}\wedge\neg\texttt{train} \rightarrow \langle\, \text{If}\,\neg\texttt{ticket}\ \text{then}\ \texttt{buy\_ticket};\texttt{enter}\,\rangle\, (\texttt{at\_platform}\wedge\neg\texttt{pay}_{10}).$$

**Regimentation.** It is not possible for an agent to enter the platform and embark the train without a ticket, i.e.:

$$\neg\texttt{at\_platform} \wedge \neg\texttt{train} \wedge \neg\texttt{ticket} \rightarrow [\texttt{enter};\texttt{embark}]\bot$$

Note that there is only one passenger agent involved in the example program. For this property, we assume that the passenger's behavior is $\texttt{enter};\texttt{embark}$. Note also that:

$$\bigcup \mathit{interleaved}(\{\texttt{enter};\texttt{embark}\}) = \texttt{enter};\texttt{embark}.$$

Below is the proof of the regimentation property above with respect to the multi-agent system program with norms that implements the train station with one passenger.

*Proof.* First, axiom *Ax Frame* using the specification of the *enter* action (with pre-condition $\{\texttt{-at\_platform}\}$ and post-condition $\{\texttt{at\_platform}\}$) gives us
(1) $-\texttt{at\_platform} \wedge -\texttt{in\_train} \wedge -\texttt{ticket} \ \rightarrow$
$[\texttt{enter}]\ \texttt{at\_platform} \wedge -\texttt{in\_train} \wedge -\texttt{ticket}$
Moreover, axiom *Ax Frame* using the specification of the *embark* action (with pre-condition $\{\texttt{at\_platform, -in\_train}\}$ and post-condition $\{\texttt{-at\_platform, in\_train}\}$) gives us
(2) $\texttt{at\_platform} \wedge -\texttt{in\_train} \wedge -\texttt{ticket} \ \rightarrow$
$[\texttt{embark}]\ -\texttt{at\_platform} \wedge \texttt{in\_train} \wedge -\texttt{ticket}$
Also, axiom *Ax Counts-as* and the specification of the second counts-as rule of the program give us
(3) $\texttt{in\_train} \wedge -\texttt{ticket} \rightarrow \texttt{viol}_{\bot}$
And axiom *Ax Regiment* together with formula (3) gives us
(4) $\texttt{in\_train} \wedge -\texttt{ticket} \rightarrow \bot$
Now, using PDL axioms together with formula (1), (2), and (4) we get first
(5) $-\texttt{at\_platform} \wedge -\texttt{in\_train} \wedge -\texttt{ticket} \ \rightarrow [\texttt{enter}][\texttt{embark}]\ \bot$
and thus
(6) $-\texttt{at\_platform}\wedge-\texttt{in\_train}\wedge-\texttt{ticket} \ \rightarrow [\texttt{enter};\texttt{embark}]\ \bot$. This completes the derivation.

## 5 Conclusions and Future Work

The paper has proposed a programming language for implementing multi-agent systems with norms. The programming language has been endowed with formal operational semantics, therefore formally grounding the use of certain social notions —eminently the notion of norm, regimentation and enforcement— as explicit programming constructs. A sound and complete logic has then been proposed which can be used for verifying properties of the multi-agent systems with norms implemented in the proposed programming language.

We have already implemented an interpreter for the programming language that facilitates the implementation of multi-agent systems without norms (see `http://www.cs.uu.nl/2apl/`). Currently, we are working to build an interpreter for the modified programming language. This interpreter can be used to execute programs that implement multi-agent systems with norms. Also, we are working on using the presented logic to devise a semi-automatic proof checker for verification properties of normative multi-agent programs.

We are aware that for a comprehensive treatment of normative multi-agent systems we need to extend our framework in many different ways. Future work aims at extending the programming language with constructs to support the implementation of a broader set of social concepts and structures (e.g., roles, power structure, task delegation, and information flow), and more complex forms of enforcement (e.g., policing agents) and norm types (e.g., norms with deadlines). Another extension of the work is the incorporation of the norm-awareness of agents in the design of the multi-agent system. We also aim at extending the framework to capture the role of norms and sanctions concerning the interaction between individual agents.

The approach in its present form concerns only closed multi-agent systems. Future work will also aim at relaxing this assumption providing similar formal semantics for open multi-agent systems. Finally, we have focused on the so-called 'ought-to-be' norms which pertain to socially preferable states. We intend to extend our programming framework with 'ought-to-do' norms pertaining to socially preferable actions.

## References

1. N. Alechina, M. Dastani, B. Logan, and J.-J.Ch Meyer. A logic of agent programs. In *Proc. AAAI 2007*, 2007.
2. P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic.* Cambridge University Press, Cambridge, 2001.
3. C. Castelfranchi. Formalizing the informal?: Dynamic social order, bottom-up social control, and spontaneous normative relations. *JAL*, 1(1-2):47–92, 2004.
4. M. Dastani. 2apl: a practical agent programming language. *International Journal of Autonomous Agents and Multi-Agent Systems*, 16(3):214–248, 2008.
5. M. Dastani and J.-J. Meyer. A practical agent programming language. In *In Proc. of ProMAS'07*, 2008.
6. V. Dignum. *A Model for Organizational Interaction.* PhD thesis, Utrecht University, SIKS, 2003.
7. M. Esteva, J.A. Rodríguez-Aguilar, B. Rosell, and J.L. Arcos. Ameli: An agent-based middleware for electronic institutions. In *Proc. of AAMAS 2004*, New York, US, July 2004.
8. D. Grossi. *Designing Invisible Handcuffs.* PhD thesis, Utrecht University, SIKS, 2007.

9. J. F. Hübner, J. S. Sichman, and O. Boissier. Moise+: Towards a structural functional and deontic model for mas organization. In *Proc. of AAMAS 2002*. ACM, July 2002.
10. A. J. I. Jones and M. Sergot. On the characterization of law and computer systems. In *Deontic Logic in Computer Science*. 1993.
11. Rosine Kitio, Olivier Boissier, Jomi Fred Hbner, and Alessandro Ricci. Organisational artifacts and agents for open multi-agent organisations: giving the power back to the agents. In *Coordination, Organizations, Institutions, and Norms in Agent Systems III*, volume 4870, pages 171–186. Springer, 2007.
12. A. Ricci, M. Viroli, and A. Omicini. "Give agents their artifacts": The A&A approach for engineering working environments in MAS. In *In proc. of AAMAS 2007*, Honolulu, Hawai'i, USA, 2007.
13. J. Searle. *The Construction of Social Reality*. Free, 1995.
14. F. Zambonelli, N. Jennings, and M. Wooldridge. Developing multiagent systems: the GAIA methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3):317–370, 2003.