Journal of Logic and Computation Advance Access published September 14, 2011

A logic for normative multi-agent programs

MEHDI DASTANI and JOHN-JULES CH. MEYER, *Information and Computing Sciences, Utrecht University, The Netherlands.*
E-mail: M.M.Dastani@uu.nl; J.J.C.Meyer@uu.nl

DAVIDE GROSSI, *Institute of Logic, Language and Computation, University of Amsterdam, The Netherlands.*
E-mail: D.Grossi@uva.nl

Abstract

Multi-agent systems are viewed as consisting of individual agents whose behaviours are regulated by an organization-oriented normative artefact. This article presents a simplified version of a programming language that is designed to implement normative 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.

Keywords: Multi-agent systems, normative systems, logic, Programming, semantics, counts-as rules.

1 Introduction

In this article, a multi-agent system is considered as consisting of a heterogeneous set of autonomous agents. Autonomy of individual agents implies that each agent pursues its own objectives and heterogeneity of agents implies that they may be developed by different designers, having different architectures, using different concepts and processes, and being implemented in different agent programming languages. The overall objectives of such multi-agent systems can be guaranteed by regulating the external (observable) behaviour of individual agents and their interactions. This amounts to assuming a normative system perspective on the to-be-designed system, i.e. to think of it in normative terms: '[…] law [and] computer systems […] may be viewed as instances of *normative systems*. We use the term to refer to any set of interacting agents whose behaviour can usefully be regarded as governed by norms' ([31], p. 276).

There have been various proposals for regulating the external behaviour of individual agents. Some of these approaches are based on coordination artefacts that regulate the behaviours of agents (or processes in general) exogenously in terms of low-level coordination concepts such as synchronization [6, 16]. Other approaches extend the notion of coordination artefacts to develop full-fledged multi-agent environments that, among other things, provide coordination facilities to individual agents [34, 35]. Yet other approaches are motivated by organizational models, normative systems, electronic institutions or a combination of normative systems and artefacts [21, 24, 30, 31, 36]. In these approaches, the behaviour of individual agents is regulated by means of norms that are enforced or regimented through monitoring 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

doi:10.1093/logcom/exr026

(e.g. roles, groups, social structures, organizations, institutions, norms) has been introduced in multi-agent system methodologies (e.g. Gaia [42]), models (e.g. OperA [19]), specification and modelling languages (e.g. $\mathcal{M}$OISE$^+$ [29] and ISLANDER [20]) and computational frameworks (e.g. AMELI [21] and $\mathcal{S}$-$\mathcal{M}$OISE$^+$ [27]).

This article collects the results from a series of articles and presentations [13, 14, 17, 18], and proposes a programming language for implementing normative artefacts and a corresponding logic to reason about normative artefact programs. The main contribution of this article is twofold. On the one hand, a simplified version of a programming language is presented that is designed to implement multi-agent systems in which the external (observable) behaviour of individual agents is regulated by means of normative artefacts. Such artefacts are implemented in terms of social concepts such as norms and sanctions, and processes such as monitoring actions performed by individual agents, evaluation of their effects, and the enforcement of sanctions, if necessary. On the other hand, we devise a logic to specify and verify properties of programs that implement norm-based artefacts.

In this article, we first briefly explain our idea of normative multi-agent systems by providing and discussing a simple but characterizing example. In Section 3, we present the syntax and operational semantics of a programming language that is designed to facilitate the implementation of norm-based multi-agent systems. This programming language allows the implementation of normative artefacts by providing programming constructs to specify norms and sanctions. In Section 4, a logic is presented that can be used to specify and verify properties of normative artefacts implemented in the presented programming language. Finally, Section 5 discusses some related works and Section 6 concludes the article and discusses some future directions of this research project.

2 Norms and multi-agent systems

Norms in multi-agent systems can be used to specify the standards of behaviour 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 just state a number of standards of behaviour in the form of a set of norms, but rather to organize the system in such a way that those standards of behaviour are actually followed by the agents. This can be achieved by regimentation or enforcement mechanisms, e.g. [24, 31].

When regimenting norms, all agents' external actions leading to a violation of those norms are made impossible. Via regimentation, the system prevents an agent from performing a forbidden action. 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 and sanctioning is that the actions that violate norms are observable by the system (e.g. fines can be issued only if the library system can detect a user being late with returning a borrowed book). Another advantage of having enforcement over regimentation is that allowing for violations contributes to the flexibility and autonomy of the agent's behaviour [11].

To better expose the kind of normative scenarios we have in mind, consider the following example. It is a variant of a somewhat 'classic' example discussed in the literature on deontic logic and computer science [31, 37].

EXAMPLE 1 (Library regulations)
The following are rules regulating the borrowing activity of a university library:

(1) Each borrower must be registered.
(2) Books should be returned by the date due.

(3) Borrowers must not exceed a given allowance of books.
(4) No books must be issued to borrowers who did not returned their books by the date due.
(5) Borrowers who have books overdue must pay a given fine.

We further assume that there are only three books in order to keep the size of the example manageable within propositional logic. It is, obviously, no essential restriction. For practical purposes, we can use Prolog like representation and reasoning schemes, which can be reduced to a propositional language by assuming finite domains and considering all possible substitutions.

Notice that Example 1 consists, first of all, of one obligation for each borrower to be registered, then one obligation to return the borrowed book on time and of a prohibition to borrow more books than a given amount. These norms specify the ideal behaviour of the system as a whole. In addition, it consists of the two so-called 'contrary-to-duties' [12, 33]. These specify the behaviour of the system in the presence of a violation of some of the previous norms. The first one states that borrowers with books overdue cannot borrow any more books or, to put it otherwise, they are considered to have already reached their maximum book allowance. In addition, the second one states an obligation for borrowers to pay a fine if they have overdue books.

2.1 *Norms as counts-as rules*

In order to formally represent normative systems, this work assumes the classificatory perspective on norms as motivated and developed in [24–26]. Accordingly, norms are statements classifying systems' states in terms of institutional terms, such as 'violation'. Borrowing the terminology advanced in [36], systems' states are specified via *brute* descriptions, e.g. 'agent i has borrowed book a'. Norms impose *institutional* descriptions upon the brute ones, e.g. 'agent i is in a violation state'. Following Example 1, we would like to express that 'states in which an agent borrows book a (in symbols, $\texttt{borrow}(a)$) without being registered (in symbols, $-\texttt{registered}$) *count as* states in which a violation occurs (in symbols, $\texttt{viol}$)'. Having the necessary literals at disposal, the logical semantics of such expressions is given via an interpretation function V yielding for each literal a subset of the systems' states domain S: $V(\texttt{borrow}(a)) \cap V(-\texttt{registered}) \subseteq V(\texttt{viol})$. In other words, counts-as expressions state the validity—in all states of the system—of the implication:

$$\texttt{borrow}(a) \wedge -\texttt{registered} \rightarrow \texttt{viol} \tag{1}$$

given an interpretation V. Section 4 will come back in more details to such logical aspects. Statements of this kind, which are often called *counts-as statements*, from the work of Searle [36], are pervasive in human institutions and organizations and constitute a basic technique of presentation of normative content. For example, if the execution of a transition α (e.g. borrow book a) by an agent from a relevant state s (e.g. where $-\texttt{registered}$ holds) always results in a state which is of a violation type given a certain counts-as statement (e.g. formula 1), then α can be considered to be prohibited in s. Counts-as statements could be quite complex and exhibit rich logical structure as shown, for instance, in [24]. In this work, however, only elementary counts-as statements, such as Formula 1, will be used for grounding programming constructs for the specification of normative systems.

Before moving to the next section, it is worth spending a few words on how this simple conception of norms relates to the literature on deontic logic. The 'counts-as reduction' of norms just sketched builds, in fact, on the tradition of the reductionistic approaches of deontic logic to alethic logics started with the work of Anderson [3–5] and Kanger [32]. In that work, the reduction of deontic statements to alethic ones is based on the intuition according to which the fact that ϕ is obligatory

means that $\neg\phi$ 'necessarily' implies a violation. The nature of the reduction lies in how this reference to a 'necessity' is formally modeled. In the present article, such necessity is assumed to be of a definitional kind, thereby requiring a universal quantification of the system's state space. In addition to the classic reductionist approach, we will also make use of several different violation constants. This, we will see, is essential in order to specify the appropriate system reaction to each one of the different sorts of violation.

2.2 Sanction rules

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 institutional facts. Typically, such brute facts are sanctions, such as fines. Notice that in human systems sanctions are usually issued by specific agents (e.g. legislators or police agents) [8–10]. 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 Programming normative artefacts

In our approach, a norm-based multi-agent system is assumed to consist of individual agents and normative artefacts that exogenously coordinate their behaviours. Since the focus of this article is on the basic concepts and processes, which are involved in normative artefact programming, as well as the formal semantics and the logic for reasoning about such artefact programs, we assume here only a single normative artefact with which agents interact. Of course, the development of practical multi-agent systems may require multiple interacting normative artefacts. In Section 6, we will explain some aspects that should be considered when multiple interacting normative artefacts are required.

In order to develop a norm-based multi-agent system, we assume individual agents being implemented in an agent programming language, not necessarily known to the normative artefact programmer. In particular, we assume an agent platform that facilitates the implementation and execution of individual agent programs. The execution of each agent program may generate a sequence of external actions. The action sequences of different agents are assumed to be interleaved by the platform scheduler. Finally, we assume the interleaved sequence of actions is observed/received by the normative artefact which is supposed to realize their effects based on the specified norms and sanctions. One of the contributions of this article is how to implement and execute a normative artefact that can enforce norms by means of regimentation and sanctioning. Most noticeably, it is not assumed that the agents are able to reason about the norms of the system.

The general architecture of such a normative multi-agent system is illustrated in Figure 1. It consists of two main components: an agent platform and a normative artefact. The agent platform facilitates the execution of individual agents and schedules their external actions. The normative artefact monitors the scheduled action sequence, realizes the effects of those actions and enforces norms by means of regimentation and sanctioning. The main process of the normative artefact, which is called Norm Enforcement Mechanism, consists therefore of monitoring agents' actions followed by enforcing the norms. This process has access to brute and institutional facts as well as the specification of actions, counts-as rules and sanction rules. In particular, when an action is performed the brute state is updated

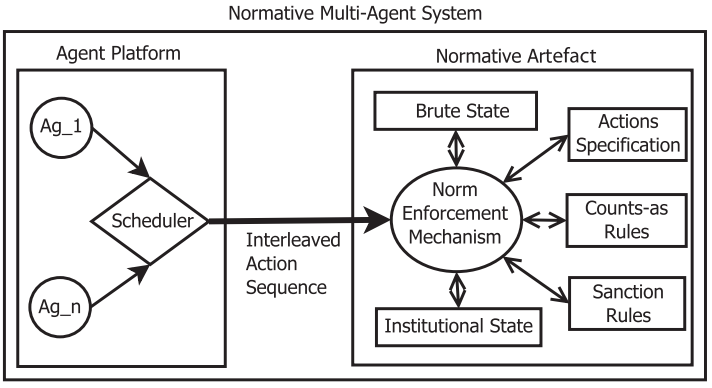

FIGURE 1. The architecture of norm-based multi-agent systems.

based on the action specification. Subsequently, the counts-as rules are used to evaluate the updated brute state to determine which norms are violated by the agents' action. Based on this evaluation and determined violations, the institutional state of the artefact is updated. Finally, the sanction rules are used to evaluate the updated institutional state to determine possible sanctions which are added to the brute state of the artefact. This process will be repeated indefinitely.

A normative artefact is to some extent comparable with a multi-agent environment in which agents perform actions to change its (brute) state. However, a normative artefact differs from a multi-agent environment in the sense that the changes that the agents can realize are specified in terms of norms and sanctions. We should immediately stress that our notion of normative artefacts is by no means an alternative for a full-fledged environment as studied in [34, 35]. We realize that a full-fledged multi-agent environment should facilitate, beside realizing the effects of actions, many different operations such as different types of sense operations. For example, our architecture does not capture the feedbacks from the artefact to the agents, which should be necessary if the normative artefact would have been an alternative for a full-fledged multi-agent environment.

In order to implement the normative artefact component of the above architecture, a programmer should specify the initial brute state of the artefact, the specification of actions, the counts-as rules and the sanction rules. Initially, there are no institutional facts; they will be generated during the run of the artefact. In the rest of this section, we present the syntax and operational semantics of a programming language to facilitate the implementation of normative artefacts. The programming language provides programming constructs to specify the effect of an agent's actions (e.g. the effects of action 'borrowing'), counts-as norms (such as rules 1–4 in Example 1), sanctions specification (e.g. rule 5 in Example 1), and the initial (brute) state of the artefact. The monitoring, regimentation and sanctioning processes constitute the interpreter of the programming language.

3.1 Syntax

As we assume that the normative artefacts determine the effects of the external actions, the programming language should provide constructs to specify these effects. The effect of an agent's (external) actions is specified by a set of literals that should hold in the brute state of the artefact. As external actions can have different effects when they are performed in different brute states of the artefact, we add a set of literals that function as the pre-condition of those effect.

```
N_Artefact_Prog   :=   "Facts: " <bruteFacts>
                       "Effects: " <effects>
                       "Counts-as rules: " <counts-as>
                       "Sanction rules: " <sanctions>;
<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>;
```

FIGURE 2. The EBNF syntax of the normative multi-agent programming language.

In order to specify brute and institutional states in our programming language, we introduce two disjoint sets of propositions to denote brute and institutional facts. The syntax of the normative artefact programming language is presented in Figure 2 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.

In the following example, we introduce the syntax that is used for specifying executable normative systems such as the one introduced in Example 1. This will concretely illustrate the programming language we are presenting.

EXAMPLE 2 (Specification of the library regulation normative system)
The specification of the initial state of the system, together with its effects, counts-as and sanction rules is given in Figure 3. It is worth stressing that such representation technique is, as such, nothing new from the point of view of deontic logic. In particular, it can be viewed as a simplified version of the norm-representation technique based on logic programming presented, for instance, in [37]. The present article will show, however, how such specification can be turned into programs—endowed with operational semantics—which execute the given specification.

This program specifies the behaviour of a normative artefact. The `Facts`, which implement brute facts, specifies the initial brute state of the normative artefact, i.e. `book(a)`, `book(b)` and `book(c)` are not borrowed and the agent (we assume for simplicity that there is only one agent) is not registered. The `Effects` indicate how the artefact can advance in its computation. Each effect is of the form `{pre-condition} action {post-condition}`. The second effect, for instance, means that if the agent performs a `borrow(a)` action when it does not already own book `a` and when it is registered in the library, the result is that the agent has borrowed book `a`. 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 institutional) state of the system. The first rule, for example, states that being late in returning book `a` determines a specific violation (marked by `viol`). We assume an internal clock in the system determining the occurring of the `late` literals. A different violation (marked by $\texttt{viol}_\perp$) follows instead if the agent has borrowed a book without being registered, or if it has borrowed all three books. In other words, as the fifth-to-seventh rules state, the agent has a maximum allowance of two books. The literal $\texttt{viol}_\perp$ is used to mark the violation of norms which will be regimented by the system. The operational semantics of the language ensures that the designated literal $\texttt{viol}_\perp$ can never hold during any run of the system (see Definition 2). Intuitively, rules with $\texttt{viol}_\perp$ as consequence could be thought of as placing gates blocking an agent's action.

```
Facts:              { -book(a), -book(b), -book(c), -registered }
Effects:            { -registered } register { registered }
                    { registered, -book(a) } borrow(a) { book(a) }
                    { registered, -book(b) } borrow(b) { book(b) }
                    { registered, -book(c) } borrow(c) { book(c) }
                    { book(a) } return(a) { -book(a) }
                    { book(b) } return(b) { -book(b) }
                    { book(c) } return(c) { -book(c) }
Counts_as rules:    { book(a), late(a) } ⇒ { viol }.
                    { book(b), late(b) } ⇒ { viol }.
                    { book(c), late(c) } ⇒ { viol }.
                    { book(a), -registered } ⇒ { viol⊥ }.
                    { book(b), -registered } ⇒ { viol⊥ }.
                    { book(c), -registered } ⇒ { viol⊥ }.
                    { book(a), book(b), book(c) } ⇒ { viol⊥ }.
                    { book(a), book(b) } ⇒ { allowance }.
                    { book(a), book(c) } ⇒ { allowance }.
                    { book(b), book(c) } ⇒ { allowance }.
                    { viol, allowance } ⇒ { viol⊥ }.
Sanction rules:     { viol } ⇒ { fined }.
```

FIGURE 3. Formal specification of Example 1.

The last counts-as rule also deserves some attention. It states that if a violation of the first type occurs, i.e. if the agent has an overdue book, and it has at least two books, i.e. it has reached its maximum allowance, a violation of type $\texttt{viol}_\perp$ occurs. In other words, the agent is not allowed to borrow a book if it already has an overdue one. Such violation will be made impossible by the system. 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 `viol` causes the sanction `fined`. A final remark about this example is that only one agent is assumed to interact with the artefact. When there are more agents interacting with the artefact, the atoms should be indexed by the agents' identities. Again, for practical purposes, we can use Prolog-like representation and reasoning schemes to accommodate indexes by means of variables.

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 Φ and consequent Ψ of the rule l, respectively. We consider only sets of rules such that (i) they are finite; (ii) they are such that each condition has exactly one associated consequence (i.e. all the consequences of a given condition are packed in a single set `cons`); and (iii) 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 $-p$), 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 article, sets of rules enjoying these three properties are denoted by $\mathbf{R}$. We also used $\mathbf{R}_c$ and $\mathbf{R}_s$ to denote sets of counts-as and sanction rules with such properties.

Note that in the current approach, a multi-agent system consists of only one normative artefact the agents interact with, which suggests a centralized approach. However, extending the notion of a multi-agent system to contain a multitude of normative artefacts the agents interact with does not affect the norm enforcement mechanism of the individual artefacts. Because this article concentrates on the norm enforcement mechanism of a normative artefact, we decided to adopt the assumption

of one artefact in order not to complicate matters further. Our framework can be extended with a set of normative artefacts, each of which is responsible for monitoring a specific set of actions and sanctioning the corresponding agents.

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 of normative artefact and the transitions that such configurations can undergo through transition rules. The configuration of a normative artefact consists of the brute and institutional states, the specifications of actions, the counts-as rules and the sanction rules.

DEFINITION 1 (Normative Artefact Configuration)
Let P_b and P_n be two disjoint sets of literals denoting atomic brute and institutional facts (including $viol_\perp$), respectively. The configuration of a normative multi-agent system is defined as $\langle \sigma_b, \sigma_n, \mathtt{Ac}, \mathbf{R_c}, \mathbf{R_s} \rangle$ where σ_b is a consistent set of literals from P_b denoting the brute state of multi-agent system and σ_n is a consistent set of literals from P_n denoting the institutional state of multi-agent system. $\mathtt{Ac}$, $\mathbf{R_c}$ and $\mathbf{R_s}$ are the sets of action specifications, counts-as rules and sanction rules, respectively.

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 article, we consider only the transition rule that specifies the transition of normative artefact configurations as a result of performing external actions by individual agents. Of course, the artefact configurations can also change because of other factors such as the internal dynamics of the artefact (e.g. the state of a clock changes independent of an individual agent's action). However, the transition rules to derive such transitions are out of the scope of this article. In our approach, the sets of action specifications, counts-as rules, and sanction rules do not change during executions. For this reason, in the rest of the article we will not include the corresponding components (i.e. $\mathtt{Ac}$, $\mathbf{R_c}$ and $\mathbf{R_s}$) in the normative artefact configurations and use $\langle \sigma_b, \sigma_n \rangle$ to represent a normative artefact configuration.

Before presenting the transition rule specifying the transition of the normative artefact 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 an 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 institutional facts, may cause the institutional state of a multi-agent system to become inconsistent.

Moreover, in order to define the transition rule to specify the transition of normative artefact configurations as a result of performing external actions by individual agents, we need to define an operation for updating the brute state of an artefact configuration. The function *up* determines the

effect of action α on the brute state σ_b based on its specification $(\Phi\ \alpha\ \Phi')$ as follows:

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

Finally, we use *performed*(α) to indicate that an agent has performed an external action α. We also assume that action α is indexed with the identifier of the agent that has performed it. We can now define a transition rule to derive transitions between normative artefact configurations.

DEFINITION 2 (Transition Rule for Normative Artefacts)
Let $\mathbf{R_c}$ be the set of counts-as rules, $\mathbf{R_s}$ be the set of sanction rules, and $(\Phi\ \alpha\ \Phi')$ be the specification of action α. The transition rule for the derivation of normative artefact transitions is defined as follows:

$$\frac{\begin{array}{c}\mathit{performed}(\alpha)\ \ \&\ \ \sigma_b\models\Phi\ \ \&\ \ \sigma'_b=up(\alpha,\sigma_b)\ \ \&\ \ \sigma'_n=\mathtt{Cl}^{\mathbf{R_c}}(\sigma'_b)\setminus\sigma'_b\\ \sigma'_n\not\models \mathit{viol}_\perp\ \ \&\ \ S=\mathtt{Cl}^{\mathbf{R_s}}(\sigma'_n)\setminus\sigma'_n\ \ \&\ \ \sigma'_b\cup S\not\models\perp\end{array}}{\langle\sigma_b,\sigma_n\rangle\longrightarrow\langle\sigma'_b\cup S,\sigma'_n\rangle}$$

where $\mathit{viol}_\perp$ is the designated literal for regimentation.

This transition rule captures the effects of performing an external action by an individual agent on both brute and institutional states of normative artefacts. First, the effect of α on σ_b is computed. Then, the updated brute state is used to determine the new institutional state of the system by applying all counts-as rules to the new brute state of the artefact. Finally, possible sanctions are added to the new brute state by applying sanction rules to the new institutional state of the artefact.

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 institutional 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 an institutional state σ'_n becomes inconsistent, the proposed transition rule cannot be applied because an inconsistent σ'_n entails $\mathit{viol}_\perp$. Also, note that the condition $\sigma'_b\cup S\not\models\perp$ guarantees that the brute state can never become inconsistent. Finally, it should be emphasized that the institutional state σ'_n is not based on σ_n and is always computed anew.

The execution of a normative artefact program is determined by the transition system that is generated by the above transition rule. The generated transition system consists of a set of finite or infinite paths $c_0,c_1,\ldots$, each starts at the initial configuration of the normative artefact c_0 and every transition $c_i\rightarrow c_{i+1}$ in a path is derivable from the transition rule. We would like to emphasize that the execution of a normative artefact, and thus the generated transition system, depends on the performed actions resulted from the execution of the individual agent programs. This is reflected in the transition rule by assuming a transition of an individual agent configuration through which an action is performed, i.e. by having *performed*(α) in the condition of the transition rule. The set of paths in the generated transition system corresponds with the order in which actions are performed.

4 Logic

In this section, we propose a logic to specify and verify liveness and safety properties of normative artefact programs. This logic, which is a variant of Propositional Dynamic Logic (PDL, see [7]), is in the spirit of [2] and relies on that work. It is important to note that the logic developed in [2] 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 normative artefact 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 institutional states of the normative artefacts. It is assumed that each propositional variable in P denotes either an institutional 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 $\sigma_b = \{(-)p_1, \ldots, (-)p_n : p_i \in P_b\}$ 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 σ_n is like σ_b for P_n.

Rules are pairs of conditions and consequences $(\{(-)p_1, \ldots, (-)p_n \mid (-)p_i \in X\}, \{(-)q_1, \ldots, (-)q_k \mid (-)q_i \in Y\})$ with X and Y being either σ_b or σ_n when applied in state $\langle \sigma_b, \sigma_n \rangle$. Following [24], if $X = \sigma_b$ and $Y = \sigma_n$ then the rule is called a *bridge counts-as rule*; if $X = Y = \sigma_n$ then the rule is an *institutional counts-as rule*. Moreover, 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. Note that the counts-as rules introduced in this article are more expressive as X can be $\sigma_b \cup \sigma_n$. Leaving technicalities aside, bridge counts-as rules connect brute states to institutional ones, institutional counts-as rules connect institutional facts to institutional facts and sanction rules connect institutional 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 by $\sigma_b \cup \sigma_n$ (σ_b in the case of a bridge counts-as rule and σ_n in the case of an institutional counts-as or a sanction rule), then $\mathtt{cons}_k$ is satisfied by σ_n (in the case of a bridge or institutional counts-as rule) or by σ_b (in the case of a sanction rule), respectively. States that are $\mathbf{R}$-aligned are states which instantiate the normative artefact 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 $i \in I$. 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 $\{(-)p_1 \in \sigma_b, \ldots, (-)p_n \in \sigma_b\}$, $\{(-)q_1 \in \sigma'_b, \ldots, (-)q_k \in \sigma'_b\}$ (where p's and q's are not necessarily disjoint) when $\alpha(i)$ is executed in a state with brute facts set σ_b which satisfies the pre-condition then the resulting state s' has the brute facts set σ'_b which satisfies the post-condition (including replacing p with $-p$ if necessary to preserve consistency) and it is such that *the rest of* σ'_b *is the same as* σ_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 $\{(\mathtt{prec}_1, \mathtt{post}_1), \ldots, (\mathtt{prec}_m, \mathtt{post}_m)\}$ 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 σ_b also univocally changes the institutional facts set σ_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 σ_b, which satisfies the pre-conditions, then the resulting state $\langle \sigma'_b \cup S, \sigma'_n \rangle$ is such that σ'_b satisfies the brute post-condition of $\alpha(i)$ (including replacing p with $-p$ if necessary) and the rest of σ'_b is the same of σ_b. Moreover, σ'_n is determined by the closure of σ'_b with counts-as rules $\mathbf{R}_c$, and sanctions S are obtained via closure of σ'_n with sanction rules $\mathbf{R}_s$.

4.2 Language

The language L for talking about normative artefact 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 institutional and brute states and a finite set $\mathtt{Ac}$ of agents' actions. Program expressions ρ are built out of external actions $\alpha(i)$ as usual, and formulae ϕ 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 $\alpha(i) \in \texttt{Ac}$ 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 **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 **C**, $\mathbf{R_c}$ and $\mathbf{R_s}$.

A model is a structure $M = \langle S, \{R_{\alpha(i)}\}_{\alpha(i)\in\texttt{Ac}}, V\rangle$ 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 institutional 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 $\alpha(i) \in \texttt{Ac}$, is a relation on S such that $(s, s') \in R_{\alpha(i)}$ iff for some $(\texttt{prec}_k, \texttt{post}_k) \in C(\alpha(i))$, $\texttt{prec}_k(s)$ and $\texttt{post}_k(s')$, 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 $\alpha(i)$. Secondly, 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 ρ 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` ϕ `then` ρ_1 `else` ρ_2 and `while` ϕ `do` ρ 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 which way the presented logic differs w.r.t. standard PDL. In fact, it is a conservative extension of PDL with domain-specific axioms needed to axiomatize the behaviour of normative artefact 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 $tr(X)$ corresponding to a (set of) pre- or post-condition(s) X as follows: $tr((-)p) = (-)p$ and $tr(\{\phi_1, \ldots, \phi_n\}) = tr(\phi_1) \wedge \ldots \wedge tr(\phi_n)$. 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 $(\mathtt{cond}_k, \mathtt{cons}_k)$ in $\mathbf{R_c}$:

$$tr(\mathtt{cond}_k) \rightarrow tr(\mathtt{cons}_k)$$

Ax Sanction For every rule $(\mathtt{viol}_k, \mathtt{sanc}_k)$ in $\mathbf{R_s}$:

$$tr(\mathtt{viol}_k) \rightarrow tr(\mathtt{sanc}_k)$$

Ax Regiment $\mathrm{viol}_\bot \rightarrow \bot$

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

$$tr(\mathtt{prec}_j) \wedge \Phi \rightarrow [\alpha(i)](tr(\mathtt{post_j}) \wedge \Phi)$$

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 $\mathtt{prec}_1, \ldots, \mathtt{prec}_k$:

$$\neg tr(\mathtt{prec}_1) \wedge \ldots \wedge \neg tr(\mathtt{prec}_k) \rightarrow \neg\langle\alpha(i)\rangle\top$$

where $\top$ is a tautology.

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

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 (Soundness and completeness)
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 ϕ to be proven satisfiable, such models are obtained via the Fischer-Ladner closure of the set of subformulae of the formula ϕ extended with all pre- and post-conditions of any action $\alpha(i)$ occurring in ϕ. Let $FLC(\phi)$ denote such closure. The canonical model consists of all 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, σ_b and σ_n are consistent, and no state is such that $\sigma_n \models \mathtt{viol}_\perp$. Secondly, it should be shown that the canonical model satisfies the pre- and post-conditions of actions occurring in ϕ in that: (i) no action $\alpha(i)$ is executable in a state s if none of its preconditions are satisfied by s, and (ii) if they hold in s then the corresponding post-conditions hold in s' which is accessible by $R_{\alpha(i)}$ from s. As to (i), 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 (ii), if a state s in the canonical model satisfies one of the preconditions $\mathtt{prec_j}$ of $\alpha(i)$ then $tr(\mathtt{prec_j})$ belongs to s and, by *Ax Frame*, $[\alpha(i)]tr(\mathtt{post_j})$ also does. 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 $tr(\mathtt{post_j})$ (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 $tr(\mathtt{post_j})$, its value cannot change from s to s' since, if it would, then by *Ax Frame* it would not be the case that $sR_{\alpha(i)}s'$, which is impossible. This concludes the proof. ■

4.5 *Verification*

In our approach, the verification of a normative artefact program means to check whether the program is soundly designed and implemented with respect to 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 normative artefact program. In order to achieve this goal, we need to translate the executions of a normative artefact program into a PDL program expression, i.e. we need to translate the set of paths in the transition system generated by the operational semantics into a PDL program expression. This translation is due to the fact that we aim at using theorem proving for verification.

As explained in earlier sections, the function of a normative artefact program is to realize the effect of the external actions (including regimentation) that are selected/decided by the execution of individual agent programs.[1] Therefore, an execution of a normative artefact program can be seen as executing a sequence of external actions that are generated by the executions of the individual agent programs. In our framework, we consider asynchronous execution of a set of individual agent programs such that the external actions performed in the normative artefact form an interleaved sequence of actions generated by the execution of the individual agent programs.

Let I be a finite set of agent names and $A_i = \alpha_i^1;\alpha_i^2,\ldots$ be an execution behaviour of agent $i \in I$, where $\alpha_i^j \in Ac_i$ (A_i is generated by the execution of the program that implements individual agent i).

[1]The general view is that the execution of an individual agent program creates a decision process that decides which actions to perform while the execution of the normative artefact program creates a process that realizes the action effect based on norms and sanctions.

Let $\mathit{interleaved}(\{A_i \mid i \in I\})$ be the set of all possible interleaving of action sequences A_i for $i \in I$. We would like to emphasize that in our approach the interleaving of actions is determined by the platform on which the individual agent programs are executing, i.e. $\mathit{interleaved}(\{A_i \mid i \in I\})$ is a function of the multi-agent system platform and thus not determined by the execution of the normative artefact program. The set $\mathit{interleaved}(\{A_i \mid i \in I\})$ represents all possible executions of a set of individual agent programs. Given a set of interleaved action sequences (possible behaviours of individual agent programs), an execution of a normative artefact program is the execution of one of the interleaved action sequences. Therefore, the executions of a normative artefact program can be translated as choices from the set of possible interleaved action sequences. Formally, we can translate the execution of a normative artefact program by the following PDL expression:

$$\bigcup \text{interleaved}(\{A_i | i \in I\})$$

It is not hard to see that the above PDL expression denotes exactly the set of paths in the transition system that is generated by the operational semantics, as presented in Section 2. To see this, note that the operational semantics consists of only one transition rule that determines the effect of one external action of an individual agent program. As the choice for this action is arbitrary (i.e. the individual agent action α in the antecedent of the transition rule is arbitrary), the set of paths in the transition system contains all paths consisting of arbitrary sequences of individual agent actions.

The general verification problem can now be formulated as follows. Given a normative artefact program with norms in a given initial state satisfying $\phi \in L$:

Liveness. After some execution of the program a state is reached satisfying ψ:

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

Safety. After any execution of the program a state is reached satisfying ψ:

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

Note that the PDL model, on which the verification formulae are evaluated, is defined with respect to a specific normative artefact program such that all states are aligned with respect to the involved counts-as and sanction rules. Moreover, the states in the model are connected in accordance with the specification of the external actions. The evaluation of a verification formula in such a model is then to check whether the interleaved sequences of external actions bring the normative artefact from from ϕ-states to states satisfying ψ.

The following proposition sketches a sample of safety and liveness properties concerning the normative artefact program specified in Section 1. Let A be an action sequence that is generated by the execution of an individual agent program. Note that $\bigcup \text{interleaved}(\{A\}) = A$ as the interleaving of one action sequence results in identical action sequence and the choice from the resulting single set is the same action sequence.

THEOREM 2 (Normative liveness and safety)
Let Δ be the set of effects, counts-as and sanction rules of the program given in Example 1, and let $\mathtt{tr}(\Delta)$ be its translation in dynamic logic (see Section 4.4).

Sanction follows violation. If the agent has book a overdue, then borrowing book b determines a violation of type $\mathtt{viol}_\perp$:

$$\mathtt{tr}(\Delta) \models (\mathtt{book(a)} \wedge \mathtt{late(a)} \wedge \mathtt{registered}) \rightarrow [\ \text{borrow}(b)\ ]\ \mathtt{viol}_\perp.$$

Norm compliance avoids sanction. If the agent returns the book on time it does not incur a fine:

$$\mathit{tr}(\Delta) \models (\mathtt{book(a)} \wedge \neg \mathtt{late(a)} \wedge \neg \mathtt{book(b)} \wedge \neg \mathtt{book(c)}) \rightarrow \langle\, \mathrm{return}(a) \,\rangle\, \neg \mathtt{fined}.$$

If the agent has borrowed books a and b, and they are not overdue, then the plan to 'either return a or b followed by borrowing c' can be executed without leading to a violation:

$$\mathit{tr}(\Delta) \models (\mathtt{book(a)} \wedge \neg \mathtt{late(a)} \wedge \mathtt{book(b)} \wedge \neg \mathtt{late(b)} \wedge \neg \mathtt{book(c)} \wedge \mathtt{registered}) \rightarrow \langle\, (\mathrm{return}(a); \mathrm{borrow}(c)) \cup (\mathrm{return}(b); \mathrm{borrow}(c)) \,\rangle\, \neg \mathtt{viol}_{\bot}.$$

Regimentation. If the agent is not registered then it cannot borrow book a:

$$\mathit{tr}(\Delta) \models \neg \mathtt{registered} \wedge \neg \mathtt{book(a)} \rightarrow [\, \mathit{borrow}(a) \,]\, \mathtt{viol}_{\bot}.$$

PROOF (SKETCH). The logical consequences can be easily proven semantically. As an example we provide the 'sanction follows violation' property. Suppose there is a model M and state s such that: $M,s \models \mathit{tr}(\Delta)$, $M,s \models \mathtt{book(a)} \wedge \mathtt{late(a)} \wedge \mathtt{registered}$, and there exists a state s' such that $sR_{\mathit{borrow}(b)}s'$ and $M,s' \not\models \mathtt{viol}_{\bot}$. It follows that $M,s \models \mathtt{viol}$. By tr it follows that $M,s' \models \mathtt{viol} \wedge \mathtt{book(a)} \wedge \mathtt{book(b)}$, from which we can conclude that $M,s' \models \mathtt{viol} \wedge \mathtt{allowance}$ and hence $M,s' \models \mathtt{viol}_{\bot}$, which is impossible. ■

5 Related work

In the literature on multi-agent systems, there have been many proposals for specification languages and logics to specify and reason about normative multi-agent systems, virtual organizations and electronic institutions (e.g. [1, 9, 31, 33]). How to develop, program and execute such normative systems was one of the central themes that were discussed and promoted during the AgentLink technical fora on programming multi-agent systems (see [15] for the general report of these technical fora). In this section, we discuss two approaches for specifying and implementing normative multi-agent systems.

One of the early modelling languages for specifying institutions in terms of institutional rules and norms is ISLANDER [22]. In order to interpret institution specifications and execute them, a computational platform, called AMELI [21], has been developed. This platform implements an infrastructure that, on the one hand, facilitates agent participation within the institutional environment and supports their communication and, on the other hand, enforces the institutional rules and norms as specified in the institutional specification. The key aspect of ISLANDER/AMELI is that norms can never be violated by the agents. In other words, systems programmed via ISLANDER/AMELI make only use of regimentation in order to guarantee the norms to be actually followed. Similar to some extensions of electronic institutions (cf. [23, 38]) we relax this assumption, guaranteeing higher autonomy to the agents and higher flexibility to the system. In contrast to our work, however, the norms of [21, 23, 38] all relate to actions the agents should or should not perform and ignore the issue of expressing more high-level norms concerning a state of the system that should be brought about. Such high-level norms can be used to represent *what* the agents should establish—in terms of a declarative description of a system state—rather than specifying *how* they should establish it.

Another approach concerning specification of normative multi-agent systems by means of social and organizational concepts is $\mathcal{M}$OISE$^+$ [29]. This modelling language can be used to specify multi-agent systems through three organizational dimensions: structural, functional, and deontic. In a series

of articles, different computational and programming frameworks have been proposed to implement and execute $\mathcal{M}$OISE$^+$ specifications. Examples of such frameworks are $\mathcal{S}$-$\mathcal{M}$OISE$^+$ [27, 28] and its artefact-based version ORG4MAS [30]. These computational frameworks, like ours, provide programming constructs to implement concepts that are investigated in social and organizational sciences. In contrast to ISLANDER/AMELI [21, 23, 38] but similar to our approach, these frameworks are concerned with more high-level norms that are about declarative descriptions of a state that should be achieved. Following the $\mathcal{M}$OISE$^+$ specification language, $\mathcal{S}$-$\mathcal{M}$OISE$^+$ is an organizational middleware that provides agents access to the communication layer and the current state of the specified organization. Moreover, this middleware allows agents to change the organization and its specification, as long as such changes do not violate organizational constraints. In the artefact version of this framework, ORG4MAS, various organizational artefacts are used to implement specific components of an organization such as group and goal schema. In this framework, a special artefact, called reputation artefact, is introduced to manage the enforcement of the norms.

To summarize, in the work on electronic institutions ISLANDER/AMELI norms pertain to low-level procedures that directly refer to actions, whereas $\mathcal{M}$OISE$^+$/$\mathcal{S}$-$\mathcal{M}$OISE$^+$ are concerned with more high-level norms pertaining to declarative descriptions of the system. However, $\mathcal{S}$-$\mathcal{M}$OISE$^+$ does not allow agents to violate organizational rules and norms by ensuring that they respect organizational specification. This suggests that norms in $\mathcal{S}$-$\mathcal{M}$OISE$^+$ are regimented rather than being enforced by means of sanctions. In the artefact version of this framework, ORG4MAS, the enforcement of norms is assumed to be managed indirectly through a reputation mechanism, but it remains unclear how such a reputation system realizes sanctioning. Another important issue is that the above-mentioned approaches lack a complete operational semantics that capture all aspects of normative systems, including the enforcement of norms. In our opinion, an explicit formal and operational treatment of norm enforcement is essential for a thorough understanding and analysis of computational frameworks of normative multi-agent systems. Also, the computational frameworks related to $\mathcal{M}$OISE$^+$ are not grounded in a logical system such that the soundness and properties of the programmed systems cannot be analysed through formal analyzes and verification mechanisms. The present article is an attempt to complement the above-mentioned works along these lines. Finally, it should be noted that ISLANDER/AMELI and $\mathcal{M}$OISE$^+$/$\mathcal{S}$-$\mathcal{M}$OISE$^+$ provide a variety of social and organizational concepts that we do not consider in this article. The focus of the present article is a simple programming language with direct monitoring and enforcement mechanisms for which its operational semantics can be grounded in a logic. An advantage of our approach is that implemented normative programs can be formally analyzed by means of verification techniques. Future research will focus on other social and organizational concepts.

6 Conclusions and future work

The article has proposed a programming language for implementing normative artefacts that can be used to coordinate the external behaviours of a heterogeneous set of autonomous agents. The programming language has been endowed with formal operational semantics, therefore formally grounding the use of certain social notions—particularly 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 normative artefacts implemented in the proposed programming language.

We have already implemented an interpreter for the proposed programming language [13]. Currently, we are working to connect this interpreter to a multi-agent system platform to build

multi-agent systems in which the behaviour of individual agent programs are coordinated by the normative artefact program. 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 artefacts we need to extend our framework in many different ways. Since the first version of this framework, we have worked on various extensions to include programming constructs to explicitly implement deontic concepts such as obligation and prohibition [41], roles [40] and norm change [39]. 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. groups, 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 one normative artefact running on a single machine. Future work will aim at relaxing this assumption providing similar formal semantics for distributed multi-agent systems, where different artefacts running on different machines are connected and organized to coordinate the behaviour of agents distributed on different agent platforms. For such an extension, one needs to study possible interactions between monitoring and sanctioning processes of different artefacts as well as how the brute and institutional facts of different artefacts can be exchanged/shared. Also, in this work we considered only the interleaved sequence of external actions that are performed by individual agents. We would like to extend our framework with synchronized actions of different agents. 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.

Acknowledgements

The authors would like to thank Nick Tinnemeier for his contribution to the first version of this article, and the anonymous reviewers of the Journal of Logic and Computation for their constructive comments.

Funding

Nederlandse Organisatie voor Wetenschappelijk Onderzoek (VENI grant Nr. 639.021.816 to D.G.).

References

[1] T. Ågotnes, W. van der Hoek, and M. Wooldridge. Robust normative systems. In *Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, L. Padgham, D. Parkes, J. Müller, and S. Parsons, eds, pp. 747–754. IFAAMAS/ACM DL, 2008.

[2] N. Alechina, M. Dastani, B. Logan, and J.-J. Ch. Meyer. A logic of agent programs. In *Proceedings of AAAI 2007*, R. C. Holte and A. E. Howe, eds, pp. 795–800, 2007.

[3] A. R. Anderson. The formal analysis of normative concepts. *American Sociological Review*, **22**, 9–17, 1957.

[4] A. R. Anderson. A reduction of deontic logic to alethic modal logic. *Mind*, **22**, 100–103, 1958.

[5] A. R. Anderson. The logic of norms. *Logique et Analyse*, **2**, 84–91, 1958.
[6] F. Arbab, L. Astefanoaei, F. S. de Boer, M. Dastani, J. J. Meyer, and N. Tinnermeier. Reo connectors as coordination artefacts in 2APL systems. In *Proceedings of the 11th Pacific Rim International Conference on Multi-Agents (PRIMA 2008)*. Vol. 5357 of *Lecture Notes in Computer Science*, T. D. Bui, T. V. Ho, and Q. T. Ha, eds, pp. 42–53. Springer, 2009.
[7] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.
[8] G. Boella and L. van der Torre. Game specification in normative multiagent system: the trias politica. In *Proceedings of IAT'04*, pp. 504–508. IEEE, 2004.
[9] G. Boella and L. van der Torre. Substantive and procedural norms in normative multiagent systems. *Journal of Applied Logic*, **6**, 152–171, 2008.
[10] G. Boella, L. van der Torre, and H. Verhagen. Introduction to normative multiagent systems. *Computational and Mathematical Organization Theory*, **12**, 71–79, 2006.
[11] C. Castelfranchi. Formalizing the informal?: dynamic social order, bottom-up social control, and spontaneous normative relations. *JAL*, **1**, 47–92, 2004.
[12] R. M. Chisholm. Contrary-to-duty imperatives and deontic logic. *Analysis*, **23**, 33–36, 1963.
[13] M. Dastani. Normative multi-agent organizations. In *Proceedings of Engineering Societies in the Agents' World (ESAW 2009)*. Vol. 5881 of *LNAI*, H. Aldewereld, V. Dignum, and G. Picard, eds, pp. 247–249. Springer, 2009.
[14] M. Dastani. Normative multi-agent programs and their logics. Presentation at Dagstuhl seminar 08361, 2008.
[15] M. Dastani and J.J. Gomez-Sanz. Programming multi-agent systems technical forum: General report. Available at: http://people.cs.uu.nl/mehdi/al3promas.html. (October 2005).
[16] M. Dastani, F. Arbab, and F.S. de Boer. Coordination and composition in multi-agent systems. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05)*, F. Dignum, V. Dignum, S. Koenig, S. Kraus, M.P. Singh, and M. Wooldridge, eds, pp. 439–446. ACM, 2005.
[17] M. Dastani, D. Grossi, J.-J. Ch. Meyer, and N. Tinnemeier. Normative multi-agent programs and their logics. In *Post proceedings of the International Workshop on Knowledge Representation for Agents and Multi-Agent Systems (KRAMAS 2008)*. Vol. 5605 of *LNAI*, J.-J. Ch. Meyer and J. Broersen, eds, pp. 16–31. Springer, 2009.
[18] M. Dastani, N. Tinnemeier, and J.-J. Ch. Meyer. A programming language for normative multi-agent systems. In *Multi-Agent Systems: Semantics and Dynamics of Organizational Models*. Information Science Reference, 2009.
[19] V. Dignum. *A Model for Organizational Interaction*. PhD Thesis, Utrecht University, SIKS, 2004.
[20] M. Esteva, D. de la Cruz, and C. Sierra. ISLANDER: an electronic institutions editor. In *Proceedings of the First International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2002)*, C. Castelfranchi and W. L. Johnson, eds, pp. 1045–1052. ACM, 2002.
[21] M. Esteva, J.A. Rodríguez-Aguilar, B. Rosell, and J. L. Arcos. AMELI: an agent-based middleware for electronic institutions. In *Proceedings of the Third International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2004)*, N. R. Jennings, C. Sierra, L. Sonenberg and M. Tambe, eds, pp. 236–243. IEEE Computer Society, 2004.
[22] M. Esteva, J. A. Rodríguez-Aguilar, C. Sierra, P. Garcia, and J. L. Arcos. On the formal specifications of electronic institutions. In *Agent Mediated Electronic Commerce, The European AgentLink Perspective*. F. Dignum and C. Sierra, eds, pp. 126–147. Springer, 2001.
[23] A. Garcia-Camino, P. Noriega, and J. A. Rodriguez-Aguilar. Implementing norms in electronic institutions. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents*

and MultiAgent Systems (AAMAS 2005), F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh and M. Wooldridge, eds, pp. 667–673. ACM, 2005.
[24] D. Grossi. *Designing Invisible Handcuffs*. PhD Thesis, Utrecht University, SIKS, 2007.
[25] D. Grossi, J.-J. Ch.. Meyer, and F. Dignum. Classificatory aspects of counts-as: an analysis in modal logic. *Journal of Logic and Computation*, **16**, 613–643, 2006.
[26] D Grossi., J.-J. Ch. Meyer, and F. Dignum. The many faces of counts-as: a formal analysis of constitutive-rules. *Journal of Applied Logic*, **6**, 192–217, 2008.
[27] J. F. Hübner, J. S. Sichman, and O. Boissier. Developing organised multiagent systems using the $\mathcal{M}$OISE$^+$ model: programming issues at the system and agent levels. *International Journal of Agent-Oriented Software Engineering*, **1**, 370–395, 2007.
[28] J. F. Hübner, J. S. Sichman, and O. Boissier. $\mathcal{S}-\mathcal{M}$OISE$^+$: A middleware for developing organised multi-agent systems. In *Proceedings of the international workshop on Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems*, Vol. 3913 of *Lectute Notes in Computer Science*, O. Boissier, J. Padget, V. Dignum, G. Lindemann, E. Matson, S. Ossowski, J. Sichman, and J. Vázquez-Salceda, eds, pp. 64–78. Springer, 2006.
[29] J. F. Hübner, J. S. Sichman, and O. Boissier. $\mathcal{M}$oise$^+$: Towards a structural functional and deontic model for mas organization. In *Proceedings of the First International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2002)*, C. Castelfranchi and W. L. Johnson, eds, pp. 501–502. ACM, 2002.
[30] J. F. Hübner, O. Boissier, R. Kitio, and A. Ricci. Instrumenting multi-agent organizations with organisational artefacts and agents: giving the organisational power back to the agents. *International Journal of Autonomous Agents and Multi-Agent Systems*, **20**, 369–400, 2010.
[31] A. J. I. Jones and M. Sergot. On the characterization of law and computer systems. In *Deontic Logic in Computer Science: Normative System Specification*, J.-J. Ch. Meyer and R. J. Wieringa, eds, pp. 275–307. John Wiley & Sons, 1993.
[32] S. Kanger. New foundations for ethical theory. In *Deontic Logic: Introductory and Systematic Readings*, R. Hilpinen, ed., pp. 36–58. Reidel Publishing Company, 1971.
[33] H. Prakken and M. Sergot. Contrary-to-duty obligations. *Studia Logica*, **57**, 91–115, 1996.
[34] A. Ricci, M. Viroli, and A. Omicini. 'Give agents their artefacts': the A&A approach for engineering working environments in MAS. In *the Proceedings of the 6th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007)*, M. Huhns, O. Shehory, E. H. Durfee, and M. Yokoo, eds, pp. 14–18. IFAAMAS, 2007.
[35] A. Ricci, M. Viroli, and A. Omicini. The A&A programming model and technology for developing agent environments in mas. In *Proceedings of 5th International Workshop on Programming Multi-Agent Systems (ProMAS 2007)*, Vol. 4908 of *LNCS*, M. Dastani, A. El Fallah Seghrouchni, A. Ricci, and M. Winikoff, eds, pp. 89–106. Springer, 2008.
[36] J. Searle. *The Construction of Social Reality*. Free Press, 1995.
[37] M. Sergot. Prospects for representing the law as logic programs. In *Logic Programming*, K. L. Clark and S. Â. Tarnlund, eds, pp. 33–42. Academic Press, 1982.
[38] V. T. Silva. From the specification to the implementation of norms: an automatic approach to generate rules from norms to govern the behaviour of agents. *JAAMAS*, **17**, 113–155, 2008.
[39] N. Tinnemeier, M. Dastani, and J.-J. Ch. Meyer. Programming norm change. In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, W. van der Hoek, G. Kaminka, Y. Lespérance, M. Luck, and S. Sen, eds, pp. 957–964. IFAAMAS/ACM, 2010.

[40] N. Tinnemeier, M. Dastani, and J.-J. Ch. Meyer. Roles and norms for programming agent organizations. In *Proceedings of the Eight International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, K. S. Decker, J. S. Sichman, C. Sierra, and C. Castelfranchi, eds, pp. 121–128. IFAAMAS/ACM, 2009.

[41] N. Tinnemeier, M. Dastani, J.-J. Ch. Meyer, and L. van der Torre. Programming normative artefacts with declarative obligations and prohibitions. In *Proceedings of IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, R. Baeza-Yates, J. Lang, S. Mitra, S. Parsons, and G. Pasi, eds, pp. 145–152. IEEE Computer Society, 2009.

[42] F. Zambonelli, N. Jennings, and M. Wooldridge. Developing multiagent systems: the GAIA methodology. *ACM Transactions on Software Engineering and Methodology*, **12**, 317–370, 2003.